UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Aplicaciones complementarias a ROBOCODE que faciliten el aprendizaje de programación en escuelas secundarias
Autor: Vanessa del Carmen Aybar Rosales
Director: Lic. Claudia Queiruga
Codirector: Lic. Claudia Banchoff
Asesor profesional:
Carrera: Licenciatura en Informática (Plan 90)

Resumen

R.I.T.A. es una aplicación destinada a cualquier usuario que quiera dar sus primeros pasos en programación.
El objetivo de R.I.T.A. es lograr que cualquier persona interesada en aprender a programar pueda centrar la atención en la lógica de programación, evitando la frustración que pudiera surgir al enfrentarse a los problemas de errores sintaxis de un lenguaje de programación particular, y en su lugar motivándolo a continuar indagando y probando sobre la herramienta. En forma simultánea, el usuario final se va familiarizando con conceptos básicos de programación, programación orientada a objetos y el lenguaje Java.
El mecanismo empleado en R.I.T.A. para la creación de algoritmos es la técnica de programación en bloques. La temática de los algoritmos consiste en la programación de robots capaces de combatir en un campo de batalla. El ganador de la batalla será el robot que implementó la mejor estrategia de combate.
R.I.T.A. genera código fuente JAVA a medida que el usuario de la aplicación compone la estrategia conectando bloques, y de manera transparente brinda la posibilidad de compilar a bytecode y disparar la ejecución contra otros robots.
R.I.T.A. es una aplicación de código abierto, escrita en Java y construída integrando OpenBlocks y Robocode.

Palabras Claves

- *algoritmos*
- *lógica de programación*
- *programación en bloques*
- *programación procedural*
- *programación orientada a objetos*
- *programación orientada a eventos*
- *Java*
- *Robocode*
- *Openblocks*

Trabajos Realizados

Implementación de un ambiente de programación en bloques que genera automáticamente código Java llamado R.I.T.A., mediante la integración de otros frameworks: Openblocks y Robocode.
En el marco del proyecto de extensión "Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica", se realizaron 2 encuentros que permitieron la evaluación de R.I.T.A.. El primer encuentro se realizó con docentes de escuelas de la provincia de Buenos Aires, mientras que el segundo se realizó con alumnos seleccionados de las mismas escuelas.

Conclusiones

A partir de la evaluación realizada con docentes y alumnos de escuelas secundarias, se probó que R.I.T.A. es una aplicación útil en la introducción a la programación procedural, conceptos básicos de programación orientada a objetos y a eventos, además de permitir introducir Java como lenguaje de programación. En particular se destaca su aspecto motivador a través de la competencia, lo que impulsa al alumno a seguir usando la herramienta e indagar en los temas asociados a la programación.

Trabajos Futuros

- *Ampliar los bloques disponibles para incluir otros tipos de robots que permitan realizar estrategias con algoritmos más complejos.*
- *Extender la funcionalidad para la permitir la formación de equipos de robots (contemplado en Robocode), lo que permitirá profundizar en conceptos de programación orientada a objetos.*
- *Elaborar una guía de enseñanza complementaria a R.I.T.A. para ser incorporada como valor agregado en las netbooks del plan Conectar-Igualdad.*

Fecha de la presentación: noviembre 2012

Objetivo

R.I.T.A. tiene tres (3) objetivos:

1. Ser una herramienta útil en la enseñanza de programación, no sólo en el aprendizaje de los conceptos que obligatoriamente los alumnos deberán asimilar, sino que debe brindar además un entorno simple y didáctico que capte el interés de los alumnos, los motive a usar la herramienta y hacerlos sentir capaces de traducir sus ideas en código ejecutable en un computador.

2. Ofrecer un ambiente gráfico integrado que permita al alumno escribir código Java, no de manera explícita, sino mediante la programación en bloques.

3. Brindar una motivación al alumno para el aprendizaje de programación mediante la competencia entre robots. De ésta manera, para el alumno, el objetivo de "aprender a programar" no es la programación en sí misma, sino lograr un mejor robot que gane a los robots de los compañeros.

Agradecimientos

Quisiera agradecer a todas las personas que de una u otra manera han colaborado en la realización de esta tesis. A mi directora Lic. Claudia Queiruga y mi co-directora Lic. Claudia Banchoff respectivamente que aceptaron dirigirme y guiarme en esta tesis; a todas las personas de las distintas áreas vinculadas con el proyecto de extensión de la Facultad de Informática: *"Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica*" que participaron con muy buena predisposición en la organización de los encuentros con docentes y alumnos de las escuelas secundarias técnicas, EEST nº 2 "Ing. Emilio Rebuelto" de Berisso y la EEST nº 5 de Berazategui, en los que se evaluó R.I.T.A.. Del área de docencia a la Lic. Laura Fava, a los alumnos Matías Brown e Isabel Kimura. Del área de comunicación a la Lic. Claudia Guidone, Tomás Bergero, Florencia Zelaya, Clara Loustaunau y Pamela Farnos. De la Dirección Pedagógica a la Prof. Ana Ungaro, Prof. Felipe Maria Celeste y Prof. Soledad Gómez. Del área de diseño a DCV Ariadna Alfano.

También quisiera agradecer a todos los docentes de las dos escuelas que participaron de dichos encuentros, quienes realmente demostraron el interés no sólo por actualizarse a nivel profesional, sino por aportar su granito de arena en mejorar la educación pública.

Por último y de manera especial, a mi familia que estuvo siempre apoyándome y esperando que me reciba: a mis padres, Carmen y Julio, por su apoyo incondicional, a Fernando, que siempre se preocupó y estuvo para escuchar mis dudas, y a mis fieles compañeros de cuatro (4) patas: Rita, que estuvo a mi lado todos los días mientras yo avanzaba en ésta tesis, y el Gordo.

Tabla de Contenidos

Introducción

En base a mi experiencia como docente, tanto en los cargos asignados en la facultad como en los cursos dictados fuera del ámbito universitario, mediante proyectos de extensión que me vincularon con alumnos de nivel terciario, pude constatar que muchas veces no resulta una tarea sencilla la enseñanza de programación. Para algunos alumnos puede resultar un doble esfuerzo el aprendizaje del concepto de "estructurar una solución" y por otro lado escribirlo de un modo que sea ejecutable en algún lenguaje de programación.

Por otro lado, no todos los alumnos que egresan de una escuela encuentran interesante el mundo de la informática, o al menos la informática no logra captar su atención.

Un aspecto interesante para trabajar en las escuelas secundarias es el de la motivación como motor para el aprendizaje. Cabe mencionar la Tesis de Magíster en Informática de la Ingeniera Zulma Cataldi (realizada en la Facultad de Informática. UNLP, año 2000, acerca de la Metodología de diseño, desarrollo y evaluación de software educativo [1], donde se plantea la motivación como un aspecto muy importante en el aprendizaje y de la que rescato las siguientes citas:

- David Perkins (1995), codirector del Proyecto Zero del Centro de Investigación para el Desarrollo Cognitivo (Harvard), en su Teoría Uno afirma que "*la gente aprende más cuando tiene una oportunidad razonable y una motivación para hacerlo*".

- Alessi y Trollip (1985), consideran que existe una motivación extrínseca independiente del programa utilizado, y una intrínseca inherente en la instrucción y recomiendan criterios para su promoción, como el uso de juegos, de exploración, de desafíos, incentivación de la curiosidad del estudiante, teniendo en cuenta un balance entre la motivación y el control del programa aplicado. Las bases teóricas pueden ser provistas por alguna de las teorías de la motivación permitiendo crear desafíos, curiosidad, control y fantasía y con un diseño motivacional que mantenga la atención a través del mismo. Los estudiantes deben poder ver la utilidad de resolución de problemas.

En Argentina, a partir del año 2005 se inició un plan de modernización de las escuelas secundarias públicas, que permitió entregar a los alumnos netbooks, a través del Programa nacional "Conectar Igualdad". Las netbooks constituyen una herramienta que bien usada, podría revolucionar la educación en la Argentina. La capacitación docente no está contemplada en este plan y en definitiva son los docentes quienes diariamente acompañan a los alumnos en el proceso de enseñanza. Los docentes deberían ser capaces de incorporar en el dictado de sus clases las netbooks y además ser guía y soporte para los alumnos en cuanto a su uso.

Habiendo realizado los comentarios previos, se decidió entonces realizar como trabajo de grado una herramienta de código abierto que apueste al aprendizaje de programación de una manera relativamente sencilla y que a la vez permita estructurar la lógica de la solución de un problema, dejando en un segundo plano el lenguaje de programación al cual se traduce la solución. La herramienta debería ser un vehículo que permita alcanzar el objetivo: pensar y estructurar la solución.

La herramienta debería permitir desarrollar una solución, que en algún punto para ser ejecutable debe traducirse a un lenguaje de programación. En este aspecto se eligió

Java[2][3] como lenguaje, por ser un lenguaje moderno y muy extendido en el ámbito educativo y laboral.

Esta tesis de grado consiste en una aplicación que como su titulo lo señala busca facilitar la enseñanza durante los primeros pasos en programación: "Aplicaciones complementarias a ROBOCODE [4] que faciliten el aprendizaje de programación en escuelas secundarias". La aplicación desarrollada se llama R.I.T.A. (Robot Inventor to Teach Algorithms) y pretende ayudar al alumno en el diseño de una solución mediante la técnica de programación en bloques.

Para que este aprendizaje resulte un poco más ameno, divertido, motivador y desafiante, se pensó que el algoritmo a implementar sea la estrategia de un robot. A medida que el alumno gesta una solución interconectando bloques, R.I.T.A. genera código Java automáticamente e inclusive le permite compilarlo y ejecutarlo, es decir, el alumno puede ver su estrategia en acción, enfrentando su robot contra otro en un campo de batalla provisto por Robocode.

R.I.T.A. es el punto de partida en el diseño de la solución. Si el usuario de esta aplicación quiere seguir aprendiendo y desea escribir una estrategia mucho más personalizada y compleja, puede usar otro ambiente de desarrollo más complejo, lo importante, es que en este punto ya adquirió los conceptos básicos, y más importante aún: la iniciativa.

El objetivo de R.I.T.A. es brindar un entorno sencillo donde el alumno pueda pensar en una solución y que esta solución quede plasmada en código sin necesidad de entender –al menos en una primera instancia– los detalles propios de un lenguaje de programación.

El desarrollo de esta tesina está formado primero por un repaso de lo que es la enseñanza de programación, considerando desde conceptos básicos relacionados a algoritmos, las técnicas de programación, evaluación de software que se usa para el aprendizaje de programación y por otro lado el estado actual de la currícula en las escuelas técnicas con orientación informática.

Adicionalmente y a modo complementario se presentan 2 capítulos: Robocode, para quienes estén interesados en ahondar un poco más en el mundo de las estrategias y batallas entre robots, y OpenBlocks [5], el framework que da soporte a R.I.T.A.

Luego se presenta a R.I.T.A. como sistema, su evolución, arquitectura y forma de uso. Se presentan también dos estudios de campo realizados con docentes de las escuelas secundarias técnicas EEST Nº 2 “Ing. Emilio Rebuelto” y la EEST Nº5 de Berazategui y con alumnos de las mismas quienes pusieron a prueba el sistema.

Finalmente presento las conclusiones de este trabajo y la visión a futuro.

Capítulo 1 – Algoritmos, una breve introducción

¿Qué es un algoritmo?

Cuando enfrentamos al alumno –sea este alumno de cualquier nivel de enseñanza– ante un problema a resolver, esperamos que nos brinde una serie de acciones a realizar que nos permitan solucionar el problema. Más allá de cómo se instrumente la solución, el éxito de esta instrumentación tiene como punto de partida el esbozo inicial de la solución que planteó el alumno.

Habiendo dicho esto, podemos indicar que un algoritmo [6][7][8] es un conjunto de pasos o instrucciones que busca resolver un problema. Un algoritmo brinda una serie de pasos, que si se ejecutan correctamente, resultarán en la solución de la tarea. Más allá del ámbito informático, los algoritmos son usados diariamente, lo que ocurre es que normalmente no pensamos explícitamente en los pasos individuales que conforman el algoritmo. Por ejemplo una jornada diaria implicará: despertarse, cambiarse de ropa, desayunar, salir de casa, poner en marcha el auto, iniciar sesión en un computador, etc. Todas estas tareas son cumplimentadas mediante una serie de acciones ejecutadas paso a paso, es decir, mediante el uso de un algoritmo.

Para que un algoritmo sea válido, cada paso (o instrucción) debería ser:

- no ambiguo: la instrucción puede ser interpretada en una única manera
- ejecutable: la persona o dispositivo que ejecuta la instrucción debe saber cómo cumplimentar la instrucción sin información extra
- ordenado: los pasos de un algoritmo tienen un orden, formando así una secuencia

A modo de ejemplo, este sencillo algoritmo –que no está relacionado al ámbito informático– nos da la secuencia de pasos para la preparación de un pisco sour:

Paso 1 → Tomar la licuadora

Paso 2 → Agregar 1 medida de jugo de limón

Paso 3 → Agregar 1 medida de jarabe de goma

Paso 4 → Agregar 2 medidas de pisco

Paso 5 → Agregar hielo (1 cubetera)

Paso 6 → Tomar 1 huevo

Paso 7 → Separar la clara de la yema de huevo

Paso 8 → Agregar la clara de huevo

Paso 9 → Licuar el contenido de la licuadora

Paso 10 → Servir en vasos

Paso 11 → Agregar en cada vaso unas gotitas de Amargo de Chuncho

Podemos decir que este algoritmo cumple los requerimientos de no ambigüedad, ejecutable y orden en los pasos a seguir.

Importancia de los Algoritmos

Los algoritmos son importantes por muchas razones:

- Un algoritmo documenta el "cómo" se cumplimenta una tarea particular
- Si un algoritmo está bien escrito, puede ser usado no sólo para cumplir una tarea simple sino un grupo de tareas relacionadas.
- La existencia de un algoritmo significa que la tarea puede ser potencialmente automatizada.

Algunos puntos a tener en cuenta acerca de los algoritmos son los siguientes:

- Para cualquier tarea que no sea trivial, probablemente existen varios algoritmos posibles que permiten cumplimentarla
- Un algoritmo indica cómo cumplir una tarea. Para poder ser llevado a cabo, se necesita entender por qué se realizan los pasos. Si bien este entendimiento no es un problema cuando un algoritmo es ejecutado por un computador, los algoritmos complejos frecuentemente tienen asociada documentación acerca de cómo funcionan.
- Algunos algoritmos son más eficientes que otros. La ejecución de un algoritmo requiere cierta cantidad de tiempo. Si un algoritmo es usado frecuentemente, su eficiencia es una cuestión crítica. Dependiendo de esta criticidad, deberá realizarse un balance entre esfuerzo y tiempo de ejecución.
- Los algoritmos deben ser "mantenibles". En general, los programas que son usados por años requieren modificaciones para adaptarse a cambios en los requerimientos de las tareas. De hecho, en el desarrollo de software en general, lo más costoso termina siendo el mantenimiento, alrededor del 75% del proceso de desarrollo de software.

En conclusión, al escribir un nuevo algoritmo, determinaremos la secuencia de pasos que resuelve un problema e idealmente buscaremos que nuestro algoritmo posea un tiempo de ejecución razonable, si es óptimo mucho mejor, y deberá estar bien documentado, de modo que en un futuro, sea fácilmente mantenible.

Capítulo 2 – Evaluación y revisión de herramientas relacionadas al software educacional

Habiendo dado una breve introducción acerca de lo que representan los algoritmos, veremos algunas de las herramientas que nos permiten plasmarlos, algunas más didácticas que otras, normalmente esto varía en función de la edad del destinatario de la aplicación.

Desde la aparición de LOGO[9] en la década del 60, se han creado diversas herramientas orientadas al software educacional. En la Figura 1 se muestra la portada de "onComputing", revista del año 1981 donde se publicaba acerca de una implementación de Logo para la computadora Texas.

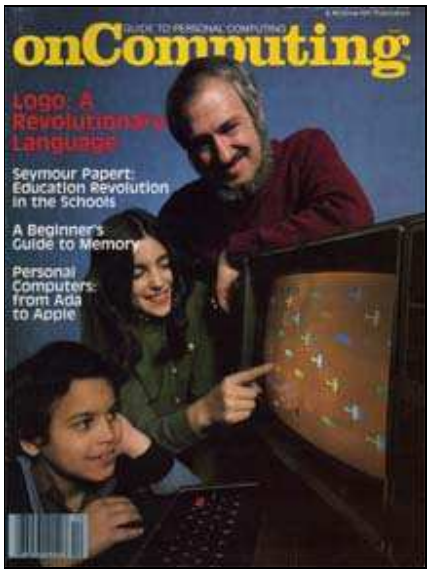


Figura 1- portada de la revista "onComputing" del año 1981

Las herramientas que nos permitirán plasmar una secuencia de acciones para lograr un objetivo pueden ser tan sencillas como un simple editor de textos, pasando por herramientas de generación de código automático, hasta sofisticados ambientes de desarrollo integrado (IDE, Integrated Development Environment) que se utilizan en la fase de desarrollo de software.

Podemos clasificar estas herramientas mediante distintos criterios, el tipo de usuario al cual están destinados, la metodología que usan para estructurar acciones bajo la forma de un programa, si están o no enfocadas a enseñar la sintaxis de un lenguaje de programación en particular, etc.

Cabe destacar que si bien muchas de las aplicaciones existente son de uso gratuito, no todas son de código fuente abierto.

A continuación revisamos algunos ambientes de programación existentes y ofrecemos una comparativa considerando la flexibilidad y facilidad que brindan a un programador novato.

Las herramientas están organizadas en las siguientes categorías:

- Ambientes de programación basados en texto
- Ambientes de programación gráfica basados en la definición de diagramas de flujo
- Ambientes de programación gráfica basados en nodos
- Ambientes de programación gráfica basados en la definición visual de reglas
- Ambientes de programación gráfica basados en bloques

Al final la revisión de cada categoría se incluye a modo de evaluación un cuadro comparativo considerando aspectos tales como si tiene disponibilidad como código abierto, si es de uso gratuito, cuales son los elementos que pueden indicarse como complejos en la herramientas, etc.

Ambientes de programación basados en texto

A nivel visual, este tipo de ambiente de programación es tan simple como un editor de texto, aunque incluyen un editor de código fuente, compilador y depurador.

Una de las herramientas más simples que ejemplifica a esta categoría es Robomind, pensado inclusive para niños. Otras herramientas son JEdit o BlueJ para programación en Java.

En esta misma categoría se ubica Eclipse, sin embargo posee más opciones que las dos herramientas mencionadas previamente. A continuación una breve descripción de cada uno de estos ambientes.

RoboMind	

RoboMind[10] es un ambiente de programación cuya temática son los Robots. Esta herramienta es open source y fue desarrollada en la Universidad de Amsterdam. Su principal característica es la simplicidad ya que está orientada a programadores novatos o niños pequeños. Como consecuencia las opciones de la aplicación son bastante limitadas.

RoboMind permite programar de manera visual y en un lenguaje propio llamado ROBO. El usuario puede incluso editar el código fuente ROBO y RoboMind realiza el chequeo en compilación. Esto resulta bastante sencillo dado que la cantidad de instrucciones es muy limitada.

Los usuarios pueden visualizar a la vez el código ROBO y cómo reacciona el Robot ante cada sentencia que está siendo ejecutada. La Figura 2 muestra la interfaz de usuario del Robomind.

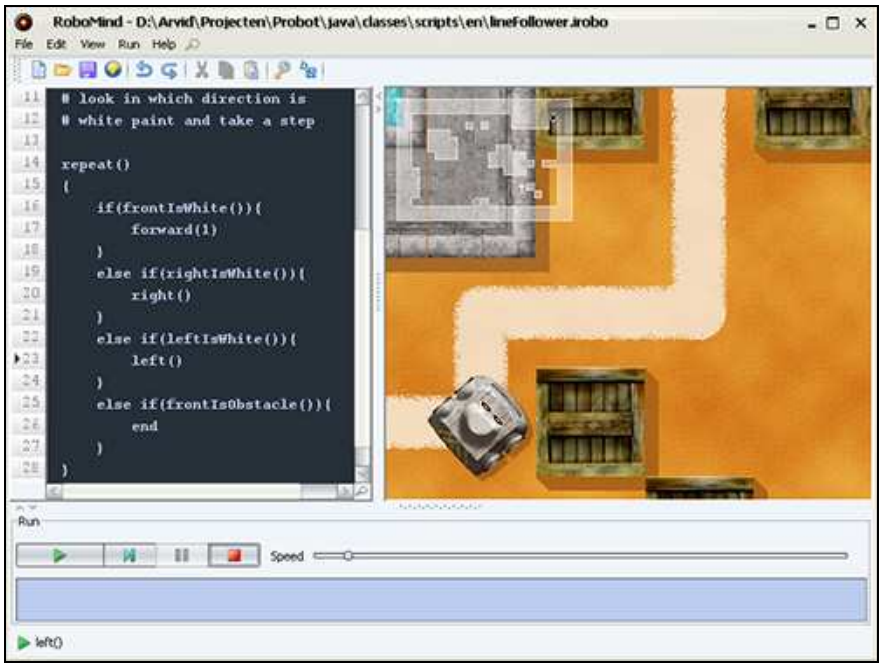


Figura 2 – RoboMind

Características del editor de texto

Todos los scripts ROBO son archivos de texto plano. Eso quiere decir que se puede usar cualquier editor de texto (como el Block de Notas de Windows o el gEdit de Linux) para escribir los programas y luego cargarlos en RoboMind.

El editor de texto integrado provee la siguiente funcionalidad:

- Números de línea para asegurar que se pueden encontrar errores en el programa fácilmente como muestra la Figura 3 muestra como se visualizan los errores.

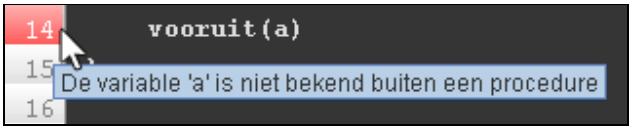


Figura 3 – Errores en RoboMind

- Los mensajes aparecen en el margen izquierdo
- "Deshacer cambios" sin límites.
- Funcionalidad "Find and Replace" (buscar y reemplazar) avanzado permitiendo expresiones regulares.
- Una flecha que indica el comando que actualmente está siendo ejecutado, la Figura 4 muestra como se visualiza la línea de código que se encuentra en ejecución.

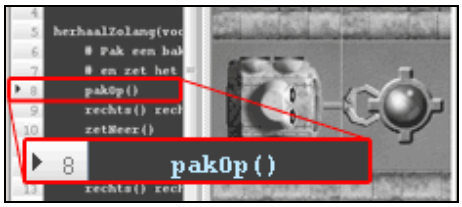


Figura 4 – Ejecución actual

Robomind tiene pocas instrucciones, lo cual resulta en una ventaja al hacer simple la programación. Además RoboMind cuenta con herramienta de "depuración", de modo que automáticamente va marcando la línea de código o un paso a paso guiado, que se ejecuta mientras se muestra los movimientos que realiza el robot.

Como desventaja de esta herramienta podemos indicar que sólo admite programación procedural. Esto limita al usuario a entender las estructuras de control básicas.

Esta aplicación cumple los requisitos de una herramienta orientada a usuarios que buscan introducirse en el mundo de la programación imperativa. Lo que se le puede criticar es el empleo de un nuevo lenguaje, cuando tal vez podría haberse usado un subconjunto de sentencias, procedimientos o métodos de algún lenguaje de programación ya existente, de modo que un usuario que completa ésta etapa introductoria, pueda seguir explorando el lenguaje de programación con el que se inició.

JEDIT

JEdit [11] fue desarrollado por Slava Pestov y posteriormente puesto a disponibilidad de la comunidad de software libre, quien quedó a cargo de su mantenimiento. JEdit está escrito en Java y se encuentra disponible como software de código abierto. La Figura 5, extraída de http://www.jedit.org, muestra la interfaz de la aplicación.

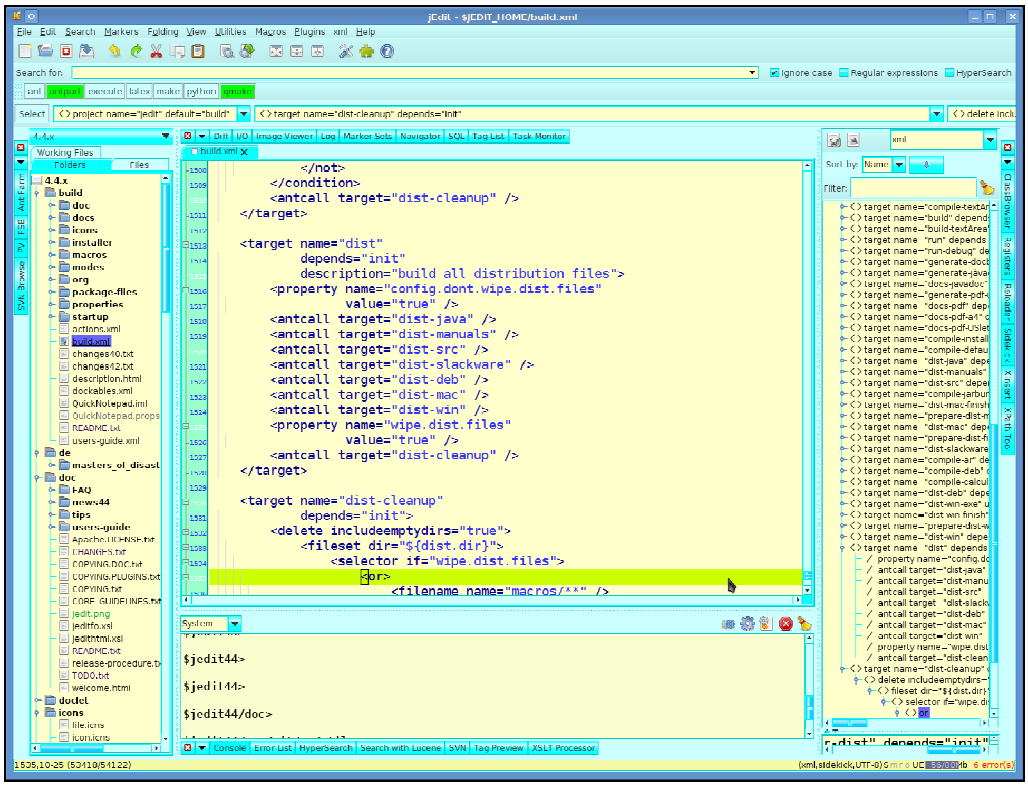


Figura 5 – interfaz JEdit

Características de JEdit:

- Se ejecuta en cualquier sistema operativo con Java 1.6+
- Soporte para más de 200 tipos de archivo
- Opciones de edición de código fuente con cantidad ilimitada de undo/redo, opción de copiar/pegar con un número ilimitado de portapapeles, facilidad de colapsar/expandir secciones de código
- Posee "markers" que le permiten recordar la posición en los archivos cuando posteriormente se vuelvan a editar
- Opciones de búsqueda y reemplazo
- Plugin FTP para abrir o guardar archivos en servidores FTP

En general brinda varias opciones básicas y sencillas de edición de código fuente, esta simplicidad hace que estos sistemas no requieran demasiada memoria y sean de rápida ejecución. Podemos indicar como desventaja que la sencillez y simplicidad de este ambiente de programación puede hacerlo parecer un tanto primitivo.

Si bien los ambientes de programación basados en texto tienen mucho potencial, en el caso de quienes se inician en la programación, puede resultar un tanto abrumador lidiar con las opciones que brinda el ambiente hasta sentirse familiarizado con el mismo.

Debemos tener en cuenta que además debe pensar en la solución a implementar y lidiar con la sintaxis del lenguaje de programación, el programador deberá sentirse cómodo en el ambiente de desarrollo.

BlueJ

BlueJ [12] fue desarrollado como parte de un proyecto de investigación universitario para la enseñanza de programación orientada a objetos a programadores novatos. Este proyecto actualmente es mantenido por La Trove University (Australia) y la University of Kent (Reino Unido), y es apoyado por Sun Microsystems. BlueJ está escrito en Java y desde el 2009 se encuentra disponible como software open source. La Figura 6, extraída de http://www.bluej.org/about/what.html muestra al editor de texto y al editor de diagrama de clases.

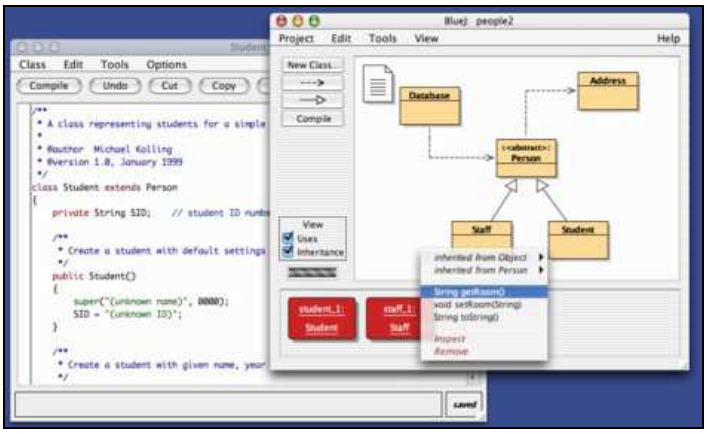


Figura 6 – Interfaz BlueJ

BlueJ proporciona:

- Diagrama de clase UML gráfico
- Editor con interfaz simple (edición de texto y gráficos)
- Compilador
- Depurador

- Creación de objetos de manera interactiva
- Llamada a objetos de manera interactiva
- Testeo interactivo

A nivel de enseñanza de programación, BlueJ resulta muy similar a JEdit. BlueJ se diferencia de JEdit cuando lo que se busca es la enseñanza de programación orientada a objetos. De hecho, en el Department of Computing Science Sam Houston State University, se hicieron pruebas de concepto acerca de cómo BlueJ podría ser realmente útil para transmitir las diferencias y ventajas de la programación orientada a objetos frente a la programación procedural, resultando en una experiencia positiva[13].

Eclipse

Eclipse [14] es un entorno de desarrollo para distintos lenguajes de programación, que brinda no sólo el ambiente de desarrollo, sino extensibilidad mediante una serie de plugins que incrementan su capacidad.

Eclipse organiza el ambiente de desarrollo en perspectivas, donde cada una está compuesta por una serie de "vistas" (paneles) en función de brindar el entorno más apropiado según la tarea que el programador lleva a cabo.

Las distintas vistas nos permiten visualizar la estructura de un proyecto, información del estado de las variables, información de las tareas pendientes (TODO), información de estructura interna de las clases (si corresponde), etc.; y por supuesto el panel principal que corresponde al código fuente. La Figura 7, extraída de http://eclipse.org muestra la interfaz de la aplicación

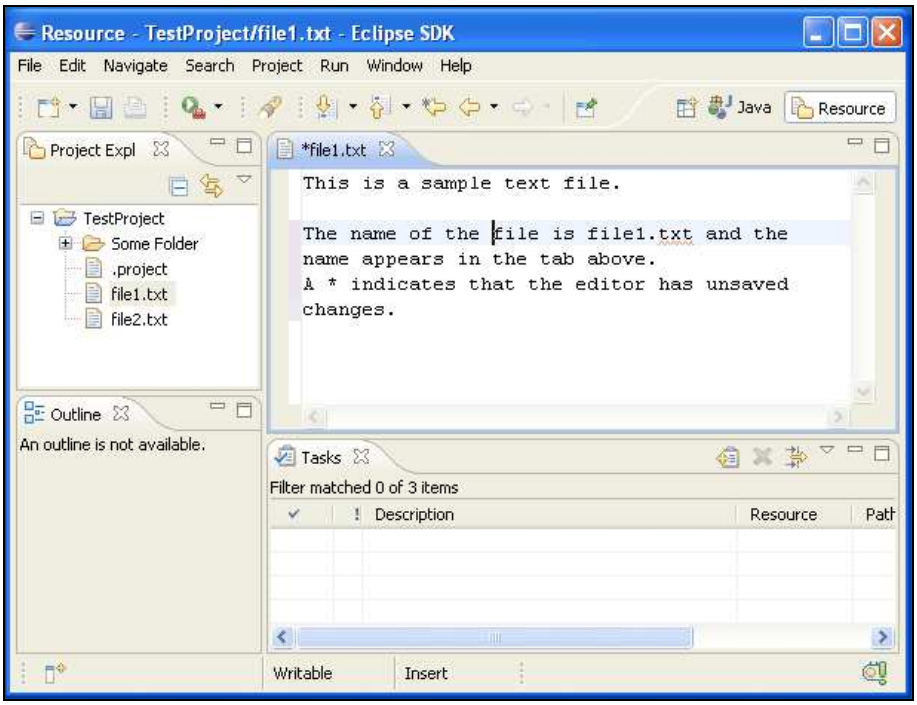


Figura 7 – Interfaz Eclipse

Acerca de los plugins que permiten extender la funcionalidad de Eclipse, muchos de estos son gratuitos y permiten entre otras cosas diseñar la GUI de una aplicación, elaborar un diagrama de clases, crear proyectos usando arquetipos, y otras funcionalidades que facilitan el trabajo del programador.

Eclipse tiene como principal ventaja ser un ambiente de desarrollo completo.

Adicionalmente, continuamente se está trabajando en una nueva versión de esta herramienta, sin mencionar el desarrollo de plugins alrededor del mismo. Sin embargo, en el caso de los programadores novatos, el hecho de brindar tantas opciones puede resultar confuso al principio del aprendijaze.

La Tabla 1 muestra un cuadro comparativo de los ambientes de programación basados en texto

Característica	RoboMind	JEdit	BlueJ	Eclipse
Open source	SI	SI	SI	SI (Eclipse SDK)
Gratuito	SI	SI	SI	SI
Elemento de complejidad de uso para programadores novatos	Idioma inglés	Orientado a Texto	Orientado a Texto	Orientado a Texto Muchas opciones
Nivel de Actividad	ALTO	ALTO	ALTO	ALTO

Complementos	NO	SI	SI	SI
Enseñanza de Orientación a Objetos	NO	NO	Mediante diagramas de clases	Mediante diagramas de clase usando algún plugin

Tabla 1 – Cuadro comparativo de los ambientes de programación basados en texto

La característica de ser orientado a texto agrega como complejidad la propensión a errores, teniendo en cuenta que los usuarios de la aplicación son programadores novatos que pueden sentirse un tanto frustrados al no poder concretar sus objetivos: ver su programa en funcionamiento.

Ambientes de programación gráficos basados en la definición visual de reglas

Este tipo de ambiente de programación consiste en poner al usuario frente a una aplicación limitada que le brinda personajes u objetos a los cuales puede indicar algún tipo de comportamiento. La acción de asignación de comportamiento se realiza de manera visual y genera reglas de comportamiento o interacción.

En este grupo haremos un repaso sobre Stagecast Creator, AgentSheet y Alice.

StageCast Creator

Stagecast Creator [15] es una aplicación escrita en Java, y por tanto requiere que esté instalada la plataforma Java. Stagecast Creator es utilizado en algunas escuelas de USA –como Cedarwood School, Mandeville, Louisiana o The Harker School, California– para enseñar programación a chicos y adolescentes mediante la interacción entre actores. Este ambiente está basado en el concepto de programación por demostración, donde las reglas son creadas dando ejemplos de qué acciones deberían llevarse a cabo dada una determinada situación.

En Stagecast Creator los actores tienen una apariencia gráfica y propiedades no gráficas. Cada actor tiene una lista de reglas que determina cómo se comporta. Las reglas se crean demostrando que hace el actor en situaciones específicas. Finalmente cada regla termina siendo de la forma “antes/después”, indicando que cuando las condiciones (“antes”) se cumplen, las acciones (“después”) son realizadas.

Como un ejemplo simple, consideremos una simulación que muestra a un actor cruzando un campo y saltando sobre las rocas que encuentra en su camino. La simulación podría comenzar con la construcción del campo de juego, en este caso, una línea de iconos podría representar el pasto y algunas rocas. Si ubicamos un actor en el campo de juego, podríamos hacer doble click y abrir un editor de reglas. El editor de reglas comenzará mostrando las condiciones actuales, es decir, el actor parado sobre el pasto y a continuación se deberá especificar las “acciones” posibles, por ejemplo, moverse a la derecha. El resultado final de esta simulación puede verse en la Figura 8, extraída del sitio web de StageCast [15].

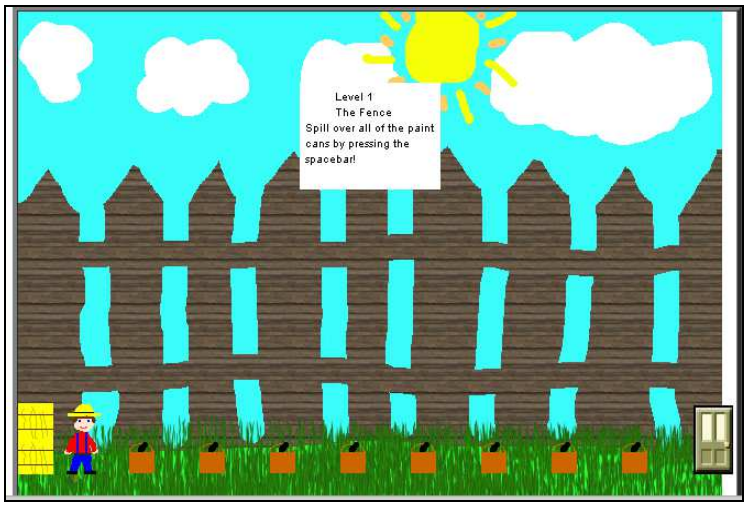


Figura 8 – Simulación realizada en StageCast Creator

Stagecast Creator no permite exportar el juego o simulación en algún lenguaje de programación en particular. Este ambiente de aprendizaje es un software propietario, sin embargo se ofrece una versión de prueba para evaluación.

La Tabla 2 muestra un resumen de los conceptos que de acuerdo a la documentación provista por el sitio web de Stagecast Creator, pueden ser aprendidos con la herramienta.

Concepto	Stagecast Creator
Objetos y herencia	Actores
Métodos	Reglas
Estructuras de control	Tests del tipo "And-if " y acciones, selectors de reglas (do first, do random, do all and continue, do in turn)
Variables	Variables (actores) y variables globales
Pasaje de parámetros	Uso de variables globales
Documentación de código	Nombres de actores, variables, reglas, etc., comentarios en las reglas
Eventos	Click de Mouse, entrada por teclado
Debugging	Reglas para Testing, herramienta “Examinar”, iluminación de reglas, deshabilitación de reglas, puntos de interrupción

Tabla 2 – Conceptos aprendidos con StageCast Creator, según sus creadores

AgentSheets

AgentSheets [16] es un software educacional que permite crear juegos que puedan ser accedidos desde un entorno web.

En la versión disponible para Windows, AgentSheets se distribuye como aplicación Java, por ende requiere ser ejecutado en la plataforma Java.

AgentSheets es usado para enseñar a estudiantes de programación prácticas relacionadas a informática mediante el diseño de juegos.

La Figura 9 muestra la Interfaz que provee AgentSheets para programar las reglas.

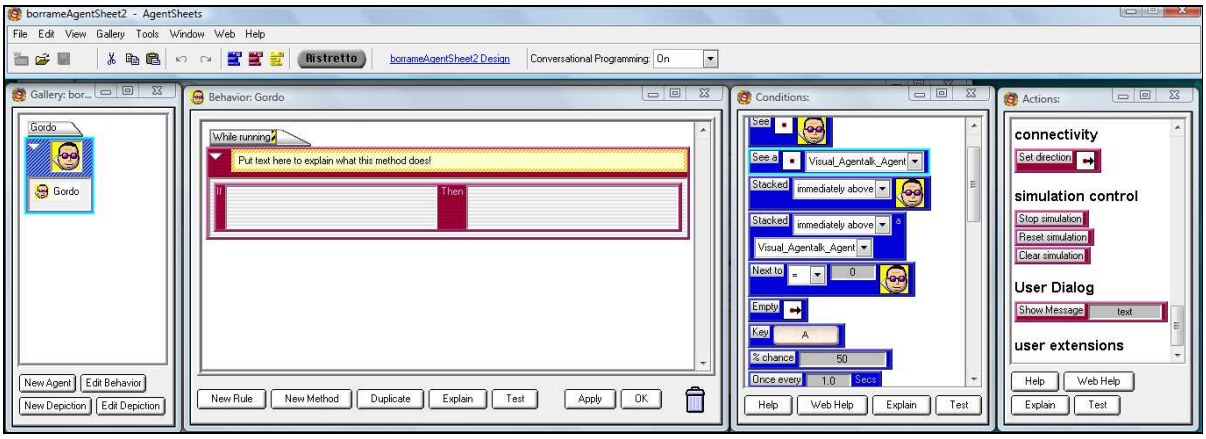


Figura 9 – Interfaz AgentSheet

AgentSheets permite construir una aplicación usando la metáfora de arrastre de componentes (drag and drop), de modo de resultar intuitivo para estudiantes sin conocimientos en programación.

Un ejemplo de aplicación simple que se puede desarrollar con AgentSheets es un simple Frogger, una vez construido puede publicarse en la web como un Applet Java.

La Figura 10 extraída de http://www.agentsheets.com/Applets/frogger1/index.html muestra un ejemplo de una aplicación Frogger programada con AgentSheets.

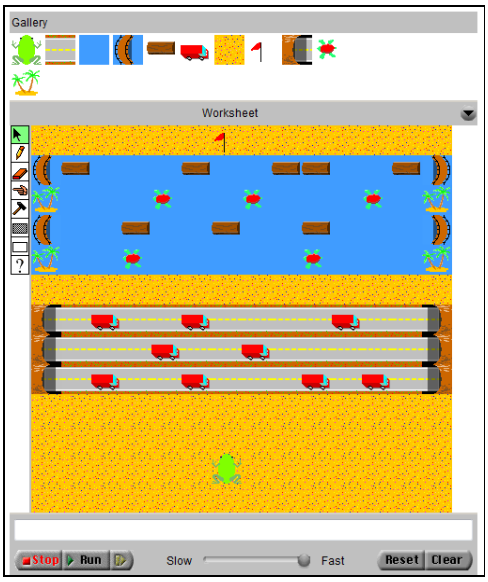


Figura 10 – Aplicación Frogger en AgentSheet

AgentSheets es usado en algunas escuelas de nivel secundario estadounidenses como Centennial Middle School o Heatherwood Elementary School [17]. Cabe destacar que es un software propietario; sin embargo, se ofrece una versión de prueba para evaluación.

Alice

Alice [18] es una herramienta desarrollada en Carnegie Mellon University que busca enseñar conceptos básicos de programación, así como la introducción de conceptos de programación orientada a objetos. Con Alice un alumno debería ser capaz de crear una historia y ver su ejecución.

La Figura 11 muestra el entorno de Alice. La figura fue extraída de la presentación de Alice realizada en el 2005.

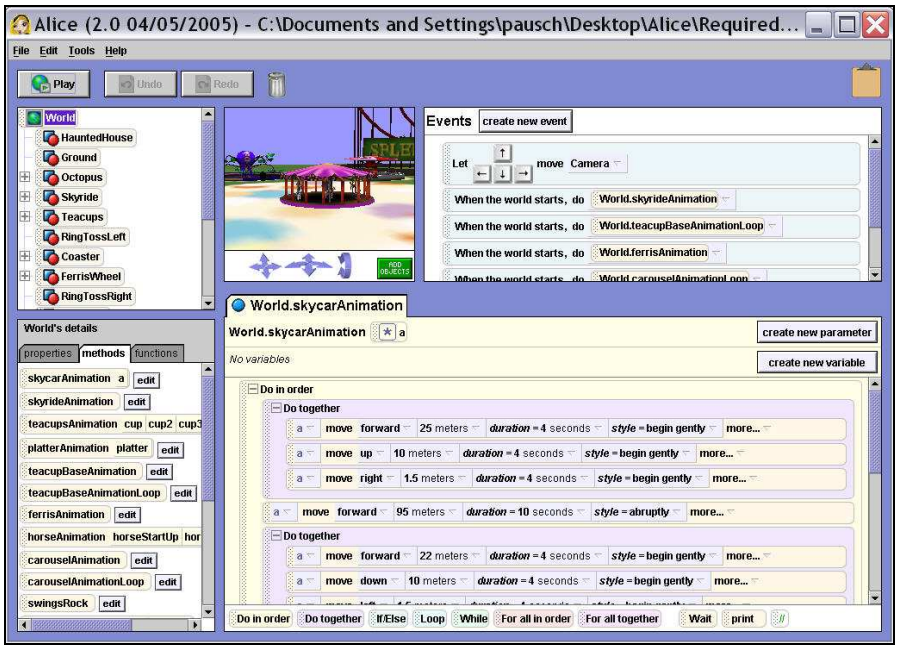


Figura 11 – Entorno Alice

En la presentación realizada por los impulsores de Alice hacia representantes de ventas de Prentice Hall [19], se describió Alice como un conjunto formado por una herramienta y un libro de respaldo "Learning to program with Alice" de Wanda Dann, Stephen Cooper, and Randy Pausch.

La Figura 12 muestra tres de los posibles actores en Alice. Cada uno de ellos representa el concepto de "clase" en el paradigma de programación orientada a objetos.

Figura 12 – Actores en Alice

La Figura 13 muestra el editor de Alice que permite al usuario generar las reglas de la aplicación.

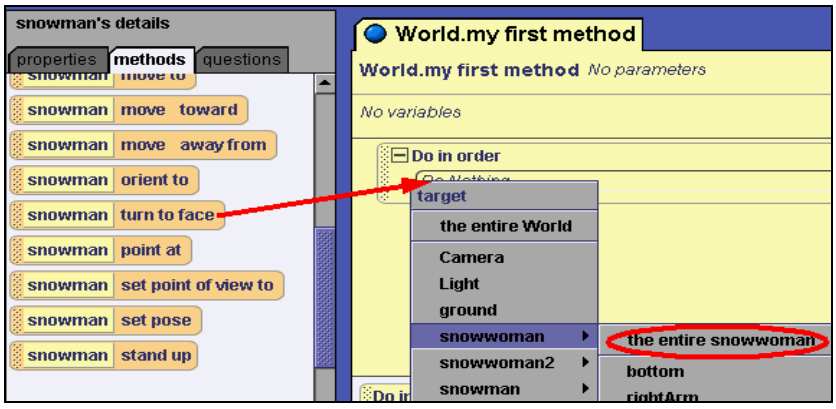


Figura 13 – Editor de Alice

Alice permite a los alumnos ubicar actores en un ambiente 3D y asignarles comportamiento. El código generado por Alice es similar a java, pero no igual. Según sus desarrolladores, debería ser simple la transición a C++, Java y VB.Net.

Alice es usado por varias escuelas de EEUU como las integrantes de Virginia Tech, San Diego State University o Community College of Philadelphia [19]. Aquí en Argentina algunas escuelas lo incluyen en la currícula, como la Asociación de Escuelas Lincoln de Argentina [20].

Como ventaja podemos destacar la relativa simplicidad para generar acciones en los personajes. Se destaca el factor motivador ya que el alumno observa el resultado final de su creación. Sin embargo, probablemente no todos los alumnos se vean atraídos por esta temática

Como desventaja cabe indicar que genera un código parecido a Java. Obviamente sería un entorno más atractivo para los docentes si el código generado fuera Java.

La Tabla 3 muestra un cuadro comparativo de los ambientes de programación gráfica basados en la definición visual de reglas

Característica	Stagecast Creator	AgentSheets	Alice
Open source	NO	NO	SI
Gratuito	NO	NO	SI
Elementos de complejidad de uso para programadores novatos	Ninguno	Idioma Inglés	Idioma Inglés
Nivel de Actividad	ALTO	ALTO	ALTO

Complementos	NO	NO	NO
Genera código	NO	NO	SI
Enseñanza de Orientación a Objetos	SI	NO	SI

Tabla 3 – Cuadro comparativo de los ambientes de programación gráfica basados en la definición visual de reglas

Ambientes de programación gráfica basados en la definición de diagramas de flujo

RAPTOR	

RAPTOR [21] (Rapid Algorithmic Prototyping Tool for Ordered Reasoning), es un software desarrollado por la US Air Force Academy escrito en una combinación de A# y C#.

RAPTOR es un ambiente de programación basado en diagramas de flujo, diseñado específicamente para ayudar a los estudiantes a visualizar sus algoritmos y evitar la carga de conocimientos relacionados a sintaxis, siendo ésta mínima. El usuario debe concentrarse en estructurar un diagrama de flujo como resolución del problema. Los programas RAPTOR son creados y ejecutados visualmente siguiendo la ejecución a través de un diagrama de flujo.

RAPTOR provee un conjunto bastante amplio de funciones (matemáticas, trigonométricas, etc.) y un inspector para visualizar los valores que van tomando las variables durante la ejecución del diagrama de flujo. Una vez escrito un programa RAPTOR, se puede generar automáticamente código escrito en JAVA, C# o Ada.

RAPTOR está orientado a usuarios que quieren introducirse al mundo de la programación considerando que el público objetivo debería ser adolescente y no niños, ya que al tener una representación de diagrama de flujo, y funciones un tanto avanzadas, se requiere un poder de comprensión mayor que el de un niño.

La Figura 14 muestra la interfaz de RAPTOR. A la izquierda las formas de diagrama de flujo disponibles, y a la derecha el programa que el usuario construye.

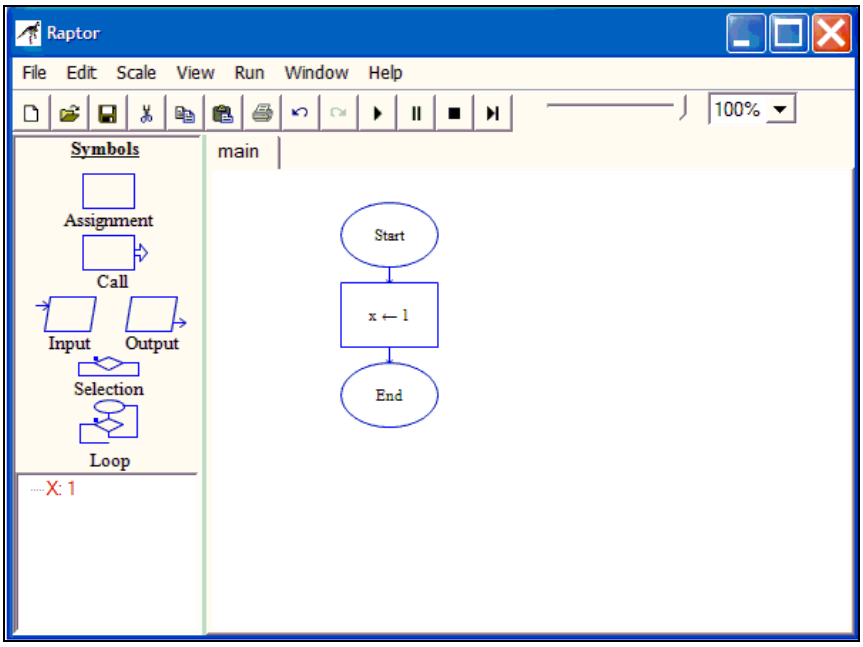


Figura 14 – Interfaz RAPTOR

Básicamente la interfaz de usuario de RAPTOR cuenta con 4 secciones:

- Barra superior, con el menú y la barra de herramientas, que permite configurar y hacer el seguimiento de la ejecución de los diagramas de flujo.
- Bloque principal/central, que muestra el diagrama de flujo que está siendo creado
- Bloque izquierdo superior, que muestra los "símbolos" que podemos arrastrar hacia nuestro diagrama de flujo
- Bloque izquierdo inferior, que muestra los valores que van adquiriendo las distintas variables a medida que se ejecuta nuestro código

Los símbolos corresponden a las acciones que pueden realizarse en Raptor:

- El símbolo de asignación, usado para dar valor a una variable.
- El símbolo de llamada, usado para hacer llamadas a procedimientos/rutinas.
- El símbolo de entrada, usado para obtener entrada desde el usuario.
- El símbolo de salida, usado para mostrar texto en la "Master Console" (consola maestra de salida)
- La estructura de selección, usada para decisiones.
- La estructura de iteración usada para repetición.

Como evaluación, podemos indicar como positivo que aun cuando parece ser una aplicación bastante simple, brinda suficiente flexibilidad al programador aún cuando la creación del algoritmo es visual.

RAPTOR además permite la invocación de procedimientos/métodos creados por el usuario de la aplicación. Lo que se le puede criticar a esta aplicación es la falta de correspondencia visual durante la ejecución entre el diagrama de flujo y el código fuente generado. Adicionalmente no se permite la edición del código fuente, todo se realiza a través de la interfaz visual.

ProgrAnimate

Progranimate [22] es una herramienta accesible a través de la web o como aplicación de escritorio, que permite la creación visual de algoritmos y generación de código. Esta herramienta está orientada a programadores novatos, y así como RAPTOR, los usuarios deberían ser al menos adolescentes. En cuanto al diseño del algoritmo, siguiendo la misma línea de RAPTOR busca una programación visual que permita al usuario concentrarse en la resolución y no en la sintaxis.

Progranimate provee una funcionalidad interesante: la sincronización entre el diagrama de flujo y el código que está siendo generado, es decir, ante la selección de una de las componentes del diagrama de flujo, se indica resaltado en gris cual es la sección de código que le corresponde.

Además, provee la capacidad de generar código en tres lenguajes: Java, Visual Basic.Net y Visual Basic 6.0.

Progranimate puede ser descargado desde internet y ejecutado vía Java Web Start o como un applet embebido en una página web. Adicionalmente puede ser instalado localmente como un JAR (Java) ejecutable en computadoras que no poseen acceso a internet.

La Figura 15 muestra la interfaz de usuario de Progranimate. De izquierda a derecha se observan: las formas disponibles para crear el diagrama de flujo, el diagrama de flujo resultante, el código fuente obtenido y la sección de visualización del estado de las variables involucradas.

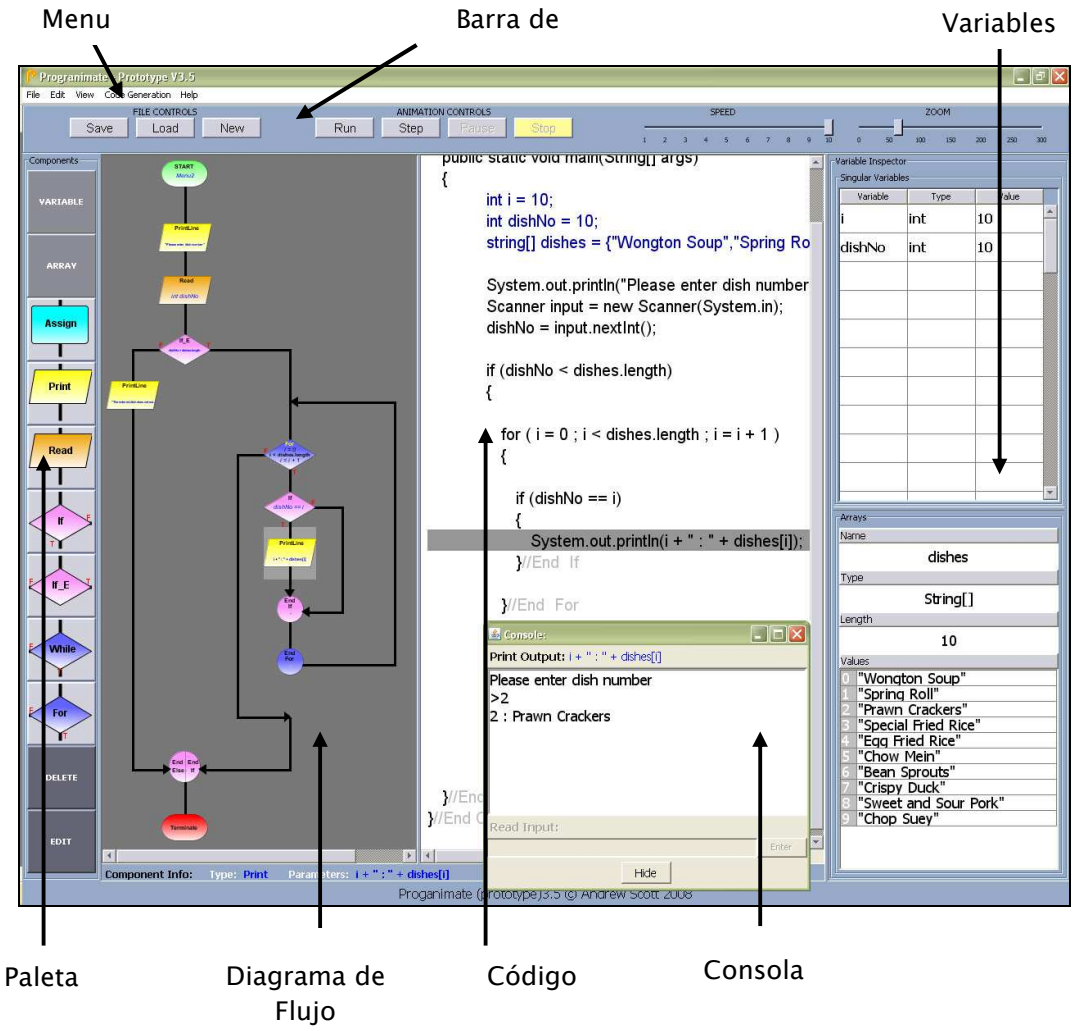


Figura 15 – Interfaz Programinate donde se muestran sus componentes

Como ventaja podemos destacar que se tiene una imagen visual completa de lo que el programa es y comprende. En Programinate no se permite la edición del código fuente, lo cual, si bien puede parecer una limitación, elimina la posibilidad de generar errores que pudieran causar confusión (consideremos que el usuario en la mayoría de los casos, podría no estar familiarizado con la sintaxis).

Acerca de la ejecución paso a paso, cabe destacar que se iluminan cada uno de los símbolos del diagrama de flujo, lo que brinda mayor comprensión de la ejecución.

Como desventaja, podemos indicar que si bien resulta bastante cómodo tener el código Java generado al lado del diagrama de flujo. En una edición inicial, prácticamente cada símbolo del diagrama de flujo queda al lado de su correspondiente código generado, pero luego, al realizar un acercamiento (zoom) esta asociación se pierde, lo cual resta claridad.

Programinate permite editar propiedades de una variable previamente ingresada. Sin embargo, no se actualizan las referencias a esta variable (no se brinda la posibilidad), de

modo que el programa no compila y el estudiante debe realizar las correcciones manualmente

Otro punto a destacar, es que el uso de Programinate implica un mínimo conocimiento de sintaxis básica necesaria. Por ejemplo, para escribir una condición, comparando un valor frente a otro, el usuario debe saber que debería usar el símbolo ==. En el caso de la sentencia "for", no permite realizar visualmente la declaración de la variable de iteración dentro del encabezado del "for", es necesario crear una variable previamente a la invocación en el "for".

En Programinate, se podrían introducir mejoras en cuanto a la ayuda al usuario, por ejemplo al momento que se genera un error cuando se modifica el código de manera manual en el diagrama de flujo, ya que considerando que el usuario no es conocedor del lenguaje, puede escribir código que genere errores, con lo cual el código no compila y al no recibir un mensaje de ayuda que clarifique la situación, se dificulta la corrección de errores

En general, Programinate es una herramienta bastante completa, pero que requiere de usuario con algún mínimo conocimiento de programación y de sintaxis de lenguajes de programación.

La Tabla 4 muestra un cuadro comparativo acerca de los ambientes de programación gráfica basados en la definición de diagramas de flujo

Característica	RAPTOR	Programinate
Open source	NO	NO
Gratuito	SI	SI
Elementos de complejidad de uso para programadores novatos	Idioma inglés	Idioma inglés
Nivel de Actividad	MEDIA	BAJO
Complementos	NO	NO
Genera código	SI	SI
Enseñanza de Orientación a Objetos	NO	NO

Tabla 4 – Cuadro comparativo acerca de los ambientes de programación gráfica basados en la definición de diagramas de flujo

Ambientes de programación basados en nodos

Si bien probablemente sea el tipo de programación con menos opciones, vale la pena mencionar dos de éstos ambientes que tienen mucho desarrollo, uno de ellos de Apple y el otro de Microsoft.

Quartz Composer

Quartz Composer [23] es un ambiente de programación visual, incluido en la suite de herramientas de desarrollo de Xcode de Apple, desarrollada para su sistema operativo OS X. Quartz Composer permite crear composiciones gráficas (como aplicaciones de efecto) y composiciones de movimiento. Todo se realiza mediante la vinculación de nodos.

La Figura 16, extraída de http://developer.apple.com/ muestra una sencilla composición.

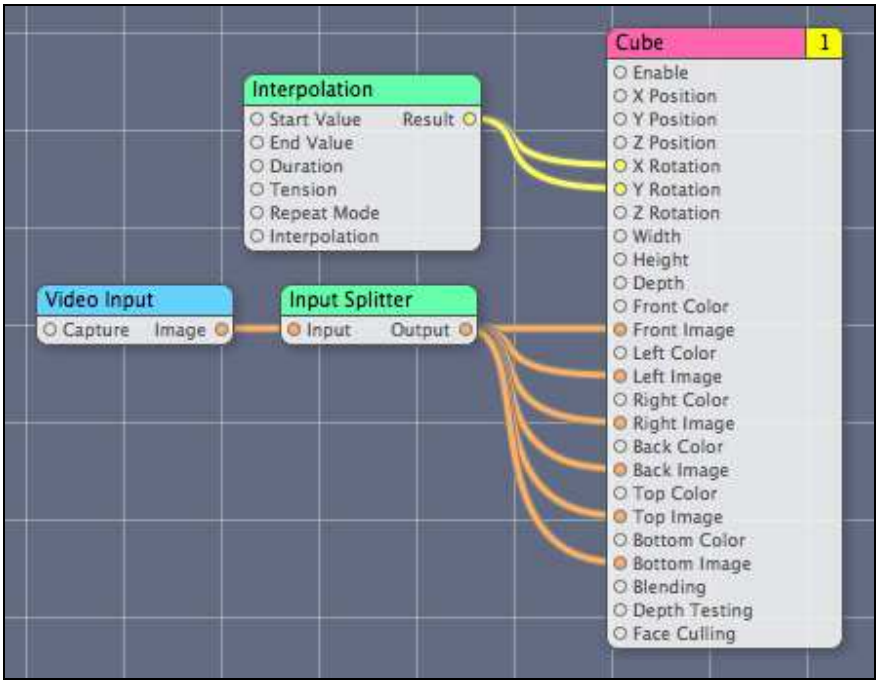


Figura 16 – Ejemplo de aplicación en Quartz Composer

Este tipo de ambiente está destinado a una audiencia que tiene una cierta noción de términos como “variables”, “paralelismo”, etc.

En función de la complejidad de la aplicación que deseamos desarrollar, el editor puede parecer un poco confuso, ya que tenemos superposición de nodos y conexiones por toda la pantalla.

Una limitante muy importante es el ambiente de ejecución requerido.

La Figura 17, extraída de E-Cell Simulation Environment 3D [24] muestra un diagrama más complejo donde se relacionan nodos.

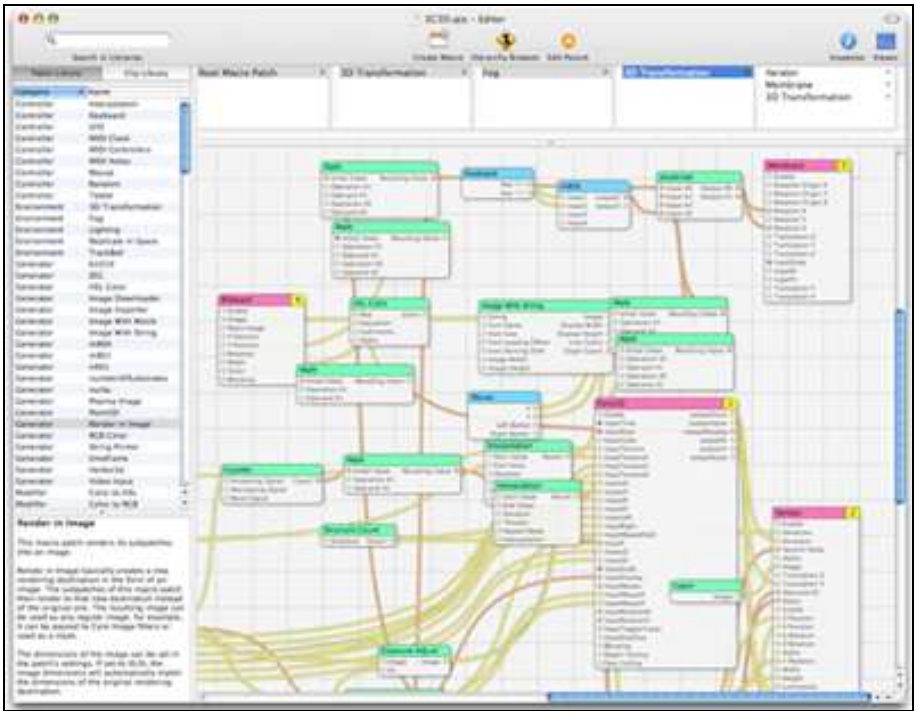


Figura 17 – Ejemplo de aplicación compleja en Quartz Composer

Si bien la interface resulta bastante agradable, el uso de Quartz Composer es bastante específico, pero se presenta como modelo que podría seguirse si se quisiera implementar una herramienta para programación en general basada en nodos.

Microsoft Robotics Developer Studio (MRDS)

MRDS [25] es una de las herramientas más completas para generación de código y cuya temática también es la programación de robots, éstos pueden ser virtuales o reales. Adicionalmente abarca aspectos de manejo de concurrencia y comunicación. Es necesario contar con la plataforma .Net para su ejecución.

Esta herramienta está orientada a todos aquellos que están interesados en robótica y pretende evitar los aspectos finos de la programación. No pareciera orientado a enseñar programación, sino a programar sin preocuparse por la sintaxis. De hecho, la aplicación tiene muchas opciones disponibles, lo cual resulta positivo para usuarios avanzados o al menos ya con conocimientos de informática; sin embargo, para un usuario novato puede parecer una aplicación bastante compleja.

MRDS se compone de Microsoft Visual Programming Language Express Edition (VPL), un ambiente de programación gráfico orientado al flujo de datos para escribir aplicaciones CCR/DSS (Concurrency and Coordination Runtime/ Decentralized Software Services). Basicamente VPL controla el flujo de datos y qué componentes son ejecutadas en función de los datos que ingresan. Las aplicaciones son composiciones de servicios los cuales son creados simplemente arrastrando y soltando componentes en un área de trabajo (canvas) y conectándolos basados en las dependencias entre los datos.

Un flujo de datos VPL consiste de una secuencia conectada de *actividades* representadas como bloques con entradas y salidas que pueden ser conectadas con otros bloques de *actividad*.

La Figura 18 muestra un ejemplo de cómo se conectan los nodos en MRDS.

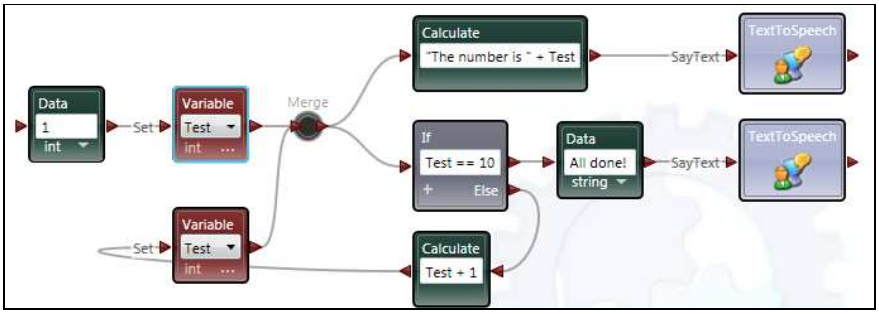


Figura 18 – Ejemplo de aplicación sencilla en MRDS, donde se realizan unos simples cálculos

Bloques de actividad tienen conexiones que representan mensajes enviados desde una actividad hacia otra

Durante la realización de pruebas con esta herramienta, fue necesario recurrir varias veces a la ayuda, lo cual demuestra que la generación visual del código no resulta del todo intuitiva.

Competición Microsoft

En el año 2008 Microsoft anunció oficialmente la competencia Robochamps basada en desarrollos sobre MRDS. Actualmente el sitio oficial http://www.robochamps.com se encuentra deshabilitado hasta la próxima convocatoria y en su lugar redirije a http://imaginecup.com/ lugar donde se desarrollan otras competencias organizadas por Microsoft.

Algunos sitios donde se puede encontrar información acerca de lo que fueron las competencias son blogs se algunos seguidores:

- http://blogs.msdn.com/angus_logan/archive/2008/04/25/robochamps-launched.aspx: blog donde se muestran algunas imágenes de la interface de RoboChamps)
- http://www.on10.net/blogs/tina/Robo-Champs-My-robot-is-bigger-then-your-robot/: sitio donde se explica en un video en qué consisten las competencias.

La Figura 19 muestra la página principal de dos de las competencias realizadas.

Figura 19 – Portal de Competencias RoboChamps

Existen varias aplicaciones que usan MRDS, a modo de ejemplo:

- Princeton University's DARPA Urban Grand Challenge (competición "driverless cars") programada enteramente en MRDS.
- MySpace usa MRDS para aplicaciones non-robotic en el back end del sitio.
- En 2008 Microsoft lanzó una competición de simulación acerca de robótica llamada RoboChamps usando MRDS, 4 desafíos están disponibles: maze, sumo, urban, y mars rover. El ambiente simulado y los robots usados por la competición fueron creados por SimplySim y la competencia fue esponsoreada por KIA Motors
- La sección de robótica y algoritmos de la competición Imagine Cup software del 2009 usó también el ambiente visual de simulación de MRDS.

Algunas consideraciones

- La aplicación tiene muchas opciones disponibles, lo cual resulta positivo para usuarios avanzados o al menos ya con conocimientos de informática, ya que para un usuario novato puede parecer una aplicación bastante compleja.
- No es multiplataforma, es requisito ejecutar la aplicación sobre plataforma .Net. Hubiera sido deseable que se hubiera ejecutado –por ejemplo – sobre Mono. (http://www.mono-project.com/)

La Tabla 5 muestra un cuadro comparativo de los ambientes de programación gráfica basados en nodos

Característica	Quartz Composer	MRDS
Open source	NO	NO
Gratuito	NO	SI
Elementos de complejidad de uso para programadores novatos	Tiene requerimientos de conceptos básicos	Tiene requerimientos de conceptos básicos
Nivel de Actividad	ALTO	MEDIO
Complementos	NO	NO
Genera código	NO	NO
Enseñanza de Orientación a Objetos	NO	NO

Tabla 5 – Cuadro comparativo de los ambientes de programación gráfica basados en nodos

Ambientes de programación basados en bloques

La idea detrás de la programación en bloques consiste en dejar de lado las cuestiones sintácticas de los lenguajes de programación, y en su lugar formar una estructura de código mediante la organización de piezas al estilo de un rompecabezas. Es decir, las formas de los bloques indican cuando es posible o no un encastre o conexión de los mismos.

Ejemplos de entornos de programación de este tipo son LogoBlocks, StarLogo TNG, AppInventor, etc.

LogoBlocks

LogoBlocks [26] es una aplicación desarrollada por Andrew Begel como miembro del Massachusetts Institute of Technology (M.I.T.) [27] y permite estructurar programas en bloques. El uso principal de esta aplicación es poder generar instrucciones que luego son enviadas a un “cricket”, el cricket no es más que una pieza de hardware que entiende un limitado número de comandos. En el caso de LogoBlocks, ésta secuencia de acciones son enviadas vía el puerto infrarrojo a un dispositivo.

La Figura 20 muestra gráficamente como se realiza la comunicación en LogoBlocks.

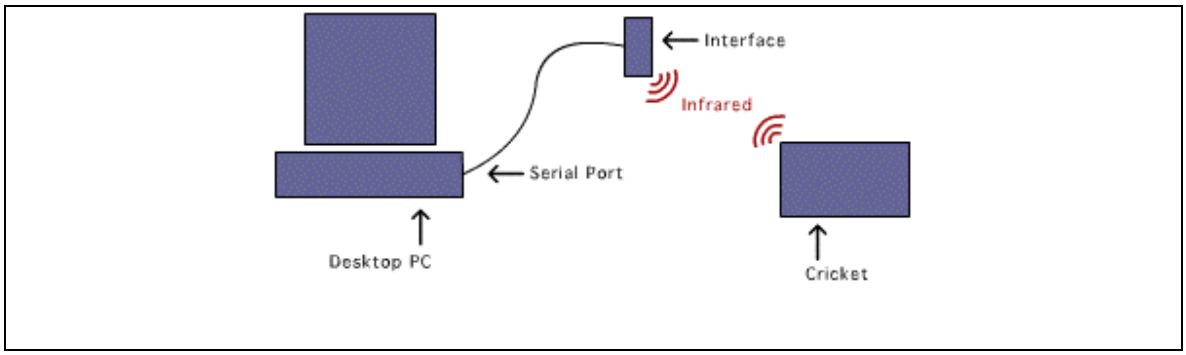


Figura 20 – Diagrama que muestra la interacción entre la PC y el Cricket

Los bloques de LogoBlocks están agrupados por colores, de modo de facilitar visualmente la lectura y relacionar la temática a la que se refiere el bloque usado. La Figura 21, extraída de la página web de introducción a LogoBlocks [28] muestra como se visualizan y forman composiciones con estos bloques.

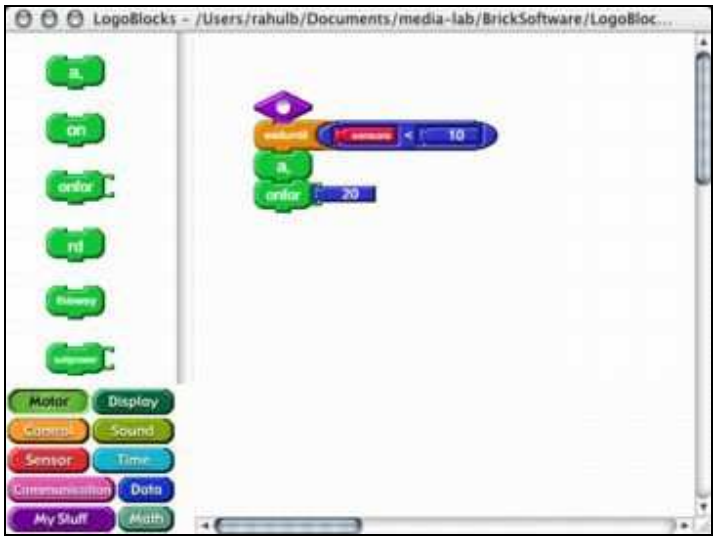


Figura 21- Composición de bloques en LogoBlocks

LogoBlocks está escrito en Java y mediante librerías de transmisión y recepción (RXTX) puede conectarse a otros dispositivos vía el puerto serial.

El uso más frecuente de LogoBlocks es la enseñanza de programación mediante la programación de chips que permitan controlar piezas de hardware relativamente simples ya que las instrucciones son limitadas.

Rescatamos de este ambiente la simplicidad al generar el algoritmo mediante la composición de bloques, como contrapartida, es limitado su uso.

StarLogo TNG

StarLogo TNG [29] es una aplicación creada por el M.I.T. en el marco del Scheller Teacher Education Program (conocido como MIT STEP) y que permite crear juegos y simulaciones 3D mediante la técnica de programación en bloques. Se podría ver como una evolución del LOGO original donde de planteaban los movimientos de una tortuga.

Los programas estructurados en StarLogo TNG pueden resultar más complejos que otros orientados a la misma temática.

La Figura 22 muestra la interfaz del ambiente de desarrollo StarLogo TNG.

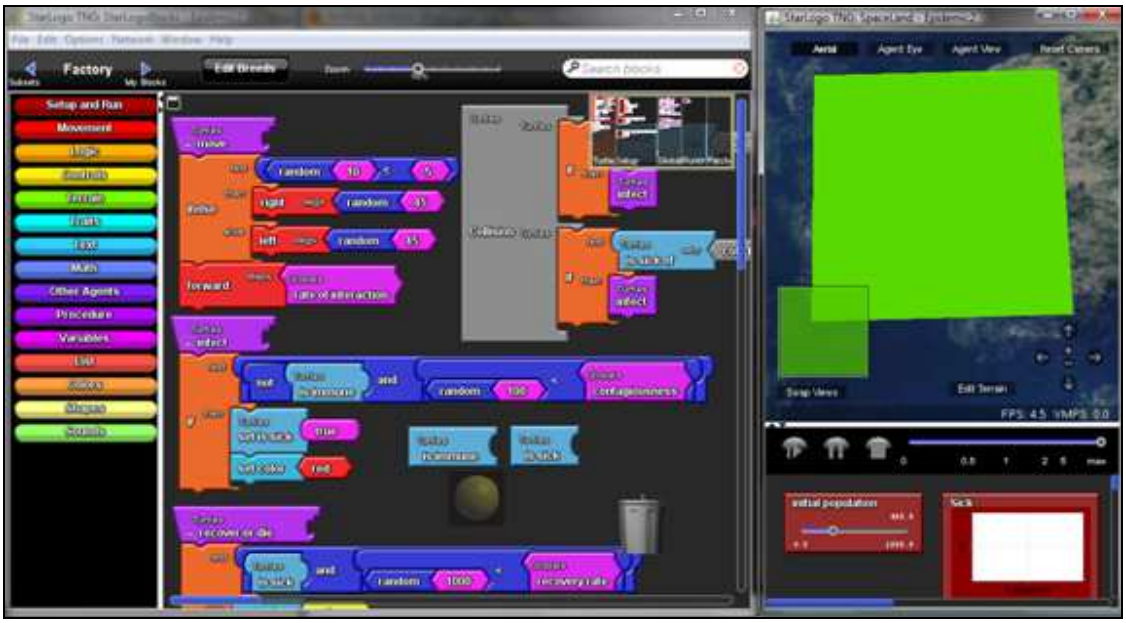


Figura 22 – Interfaz StarLogo TNG

StarLogo TNG tiene la misma flexibilidad que LogoBlocks al permitir la estructuración de bloques. Adicionalmente brinda un escenario 3D donde se visualiza la ejecución (los movimientos de una tortuga).

StarLogo TNG está escrito en Java y por tanto, para ejecutarse necesita que esté instalada la plataforma Java. Si bien la descarga de StarLogo TNG es gratuita, aún no es opensource.

App Inventor

App Inventor [30] es una aplicación originalmente provista por Google y actualmente mantenida por el MIT, específicamente por el MIT Center for Mobile Learning. Según se indica en su página de presentación, esta aplicación permite a personas que no están familiarizadas con la programación de computadoras, crear software que se ejecute en el sistema operativo Android (es decir, aplicaciones para dispositivos móviles).

La interfaz es muy parecida a la que brinda StarLogo TNG, y de hecho, el editor de bloques guarda estrecha relación con StarLogo TNG debido a que ambos usan el framework OpenBlocks, el cual permite que los usuarios puedan visualmente arrastrar y soltar los bloques para crear la aplicación Android. Esta aplicación se desarrolló durante el año 2010 por Google. En agosto de 2011, Google anunció que App Inventor sería discontinuado

como producto de Google y que se distribuiría de modo open source como parte del MIT Center for Mobile Learning.

La interacción por parte del programador con App Inventor se realiza a través de internet, vía un navegador web. La aplicación puede conectarse con un teléfono o en su defecto con un emulador. Los servidores de App Inventor almacenan los proyectos de los usuarios. La Figura 23 muestra los componentes que participan en una aplicación AppInventor.

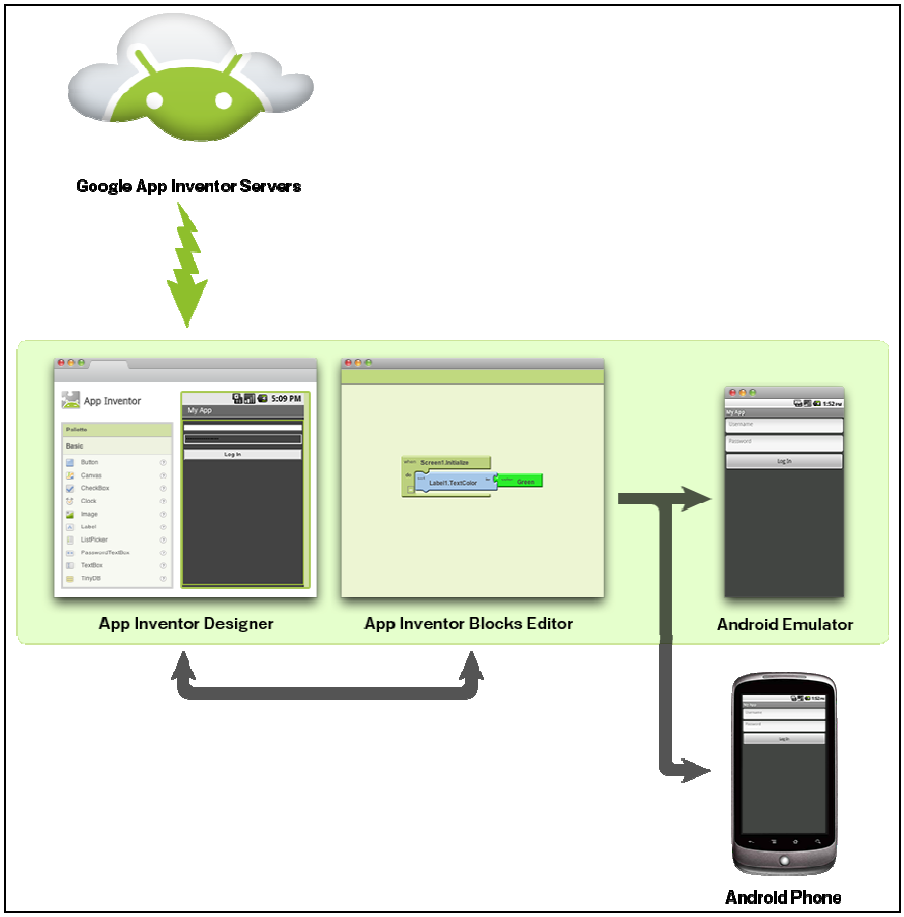


Figura 23 – Componentes de una aplicación AppInventor

Para construir una aplicación, el usuario interactúa con las siguientes aplicaciones:

- App Inventor Designer, donde se seleccionan las componentes de la aplicación.
- App Inventor Blocks Editor, donde se ensamblan los bloques que conforman el programa. Las componentes se ensamblan visualmente como piezas de un rompecabezas.
- Es posible testear la aplicación en un teléfono. Una vez testeado se empaqueta la aplicación con extensión "apk" para su distribución de modo de producir una aplicación stand-alone.

El estado actual de AppInventor es una versión Beta.

La Figura 24 muestra el entorno de trabajo App Inventor Designer, mientras que la Figura 25 muestra la estructuración de los bloques en App Inventor Blocks.

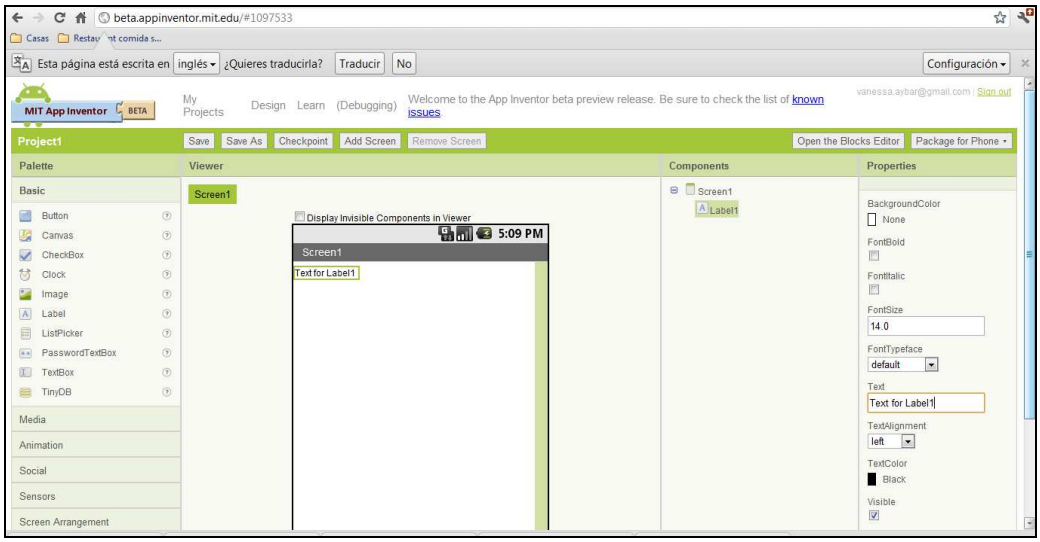


Figura 24 – Interfaz AppInventor

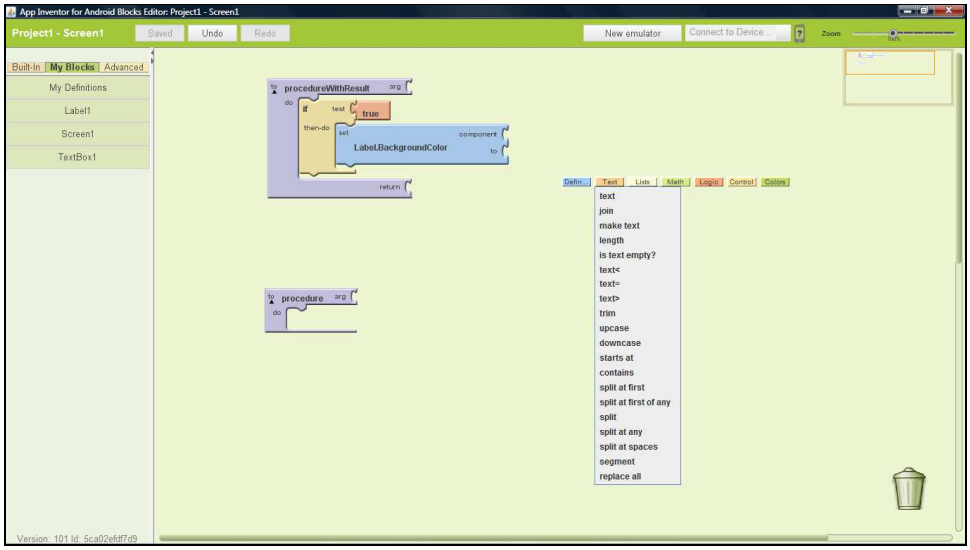


Figura 25 – Composición de bloques en AppInventor

Nuevamente, se rescata la simplicidad de este tipo de programación, a la vez se destaca la facilidad para crear aplicaciones relativamente simples que puede llegar a desarrollar un programador novato. Como contrapartida, podemos señalar que AppInventor no permite descargar el código generado, sólo el paquete final que se distribuirá como aplicación Android.

La Tabla 6 muestra un cuadro comparativo de los distintos ambientes de programación basados en bloques evaluados.

Característica	LogoBlocks	StarLogo TNG	App Inventor
Open source	NO	NO	NO
Gratuito	SI	SI	SI
Elementos de complejidad de uso para programadores novatos	Ninguno	Ninguno	Ninguno
Nivel de Actividad	BAJO	BAJO	MEDIO
Complementos	NO	NO	NO
Genera código	NO	NO	NO
Enseñanza de Orientación a Objetos	NO	NO	NO

Tabla 6 – Cuadro comparativo de los distintos ambientes de programación basados en bloques evaluados

Conclusiones acerca de las herramientas evaluadas

La idea detrás de la evaluación de las herramientas vistas fue rescatar todo aquello que puede resultar interesante en una aplicación cuyo objetivo sea el aprendizaje de la estructuración de ideas y la programación. Si bien hay una variedad muy amplia de aplicaciones y están destinadas a distinto tipo de audiencia, con algunas salvedades particulares el objetivo es el mismo.

Si bien hay herramientas muy completas, ese mismo nivel de completitud hace que sea más complejo el uso considerando como audiencia un programador novato.

A continuación algunas de las características que considero son deseables en un programa que va a ser usado por un programador novato.

- **La simpleza de la programación en bloques:** ya que la misma forma de los bloques nos indica como estructurarlos de una manera válida
- **Motivación e iniciativa de captar la atención de los usuarios de la aplicación:** Alice, LogoBlocks o StarLogo TNG permiten ver los resultados del trabajo realizado. Por otro lado, Microsoft Robotics Developer Studio plantea adicionalmente otro tipo de motivación: el reconocimiento en competencia.
- **La correspondencia visual entre la especificación de un algoritmo y el código fuente resultante:** éste es el caso de Raptor.
- **El seguimiento paso a paso de la secuencia de pasos que conforman nuestro algoritmo:** en el caso de RoboMind lo realiza de una manera muy amigable para el usuario.

Capítulo 3 – Enseñanza de Informática en las escuelas secundarias técnica de la provincia de Buenos Aires

El Plan de mejoras del Instituto Nacional de Educación Tecnológica (INET) [31], mediante la ley 26058 [32] promulgada en septiembre del 2005, permitió a las escuelas secundarias técnicas equipar su laboratorios de computación con equipamiento específico de redes, con PCs modernas. Luego el programa Conectar-Igualdad, les asignó a todos sus alumnos netbooks, también se actualizó el diseño curricular creándose la nueva escuela secundaria técnica que tiene 7 años, con 6 obligatorios y uno optativo, que tiene como terminal el título de técnico en informática (7 años) y bachiller con orientación en informática (6 año). Sin embargo todas estas mejores no fueron acompañadas adecuadamente con actualización a sus docentes, de manera de incorporar en el aula de la escuela técnica saberes que luego puedan ser retomados en la universidad para aquellos estudiantes que continúen sus estudios de informática en el nivel universitario o generar un técnico informático que pueda incorporarse en el mundo del trabajo de la tecnología.

Para abordar el tema de enseñanza de informática en escuelas secundarias técnicas de la provincia de Buenos Aires, debemos tener en cuenta que al día de hoy es casi imposible no estar conectado de alguna manera a Internet, ya sea de una manera muy básica, a través de nuestras cuentas de correo, o integrándonos a la sociedad virtual a través las redes sociales o a través de los juegos en red. Debemos reconocer que estos últimos captan fuertemente la atención de niños y adolescentes.

Hoy en día, si bien existe un diseño curricular en las escuelas acerca de los contenidos a impartir respecto de prácticas informáticas, la forma de implementarlo puede variar de una a otra. Muchas veces depende de la capacidad y la formación del docente que se hace cargo del curso.

Con respecto a la escuela primaria, algunas escuelas ofrecen un primer acercamiento a la informática mediante aplicaciones didácticas (como Alice) que no enseñan exactamente a programar sino más bien, a poder armar escenarios con personajes que interactúan, permitiendo de esta forma acercar al alumno al uso de una computadora y a la creación de una aplicación.

En la escuela secundaria técnica de la provincia de Buenos Aires, no hay un estándar acerca de un lenguaje de programación en particular, en general se enseña a usar herramientas de ofimática como el paquete OpenOffice o MS-Office, e inclusive tienen convenios con universidades, como la Universidad Tecnológica Nacional, que brinda certificación en este tipo de herramientas [33].

En conclusión, no existe un consenso acerca de los contenidos, depende de la capacidad y entusiasmo por parte del docente.

En otro nivel, tenemos a la escuela secundaria técnica. Todo este cambio en nuestra sociedad, implicó que el contenido curricular de la escuela secundaria técnica se actualice. El diseño curricular actual en la escuela secundaria técnica de la provincia de Buenos Aires responde a lo establecido en el año 2008, de acuerdo a la Ley de Educación Nacional, de Educación Provincial y de Educación Técnico Profesional y el Decreto Nº 144/08 [34]. De

esta forma, se oficializa la incorporación de tecnicaturas informáticas en la escuela secundaria técnica.

De acuerdo a lo establecido en el Decreto Nº 144/08, se estableció que la Educación Secundaria Técnica y la Educación Secundaria Agraria conforman alternativas de educación obligatoria, con siete años de duración, y constituyen unidades pedagógicas y organizativas comprendidas por una formación común y una orientada, de carácter diversificado, que responden a diferentes áreas del conocimiento. El ciclo de formación común a todas las tecnicaturas es conocido como Básico y comprende 3 años, mientras que los 4 años restantes corresponden a formación específica de cada especialidad.

Se estableció que cumplidos los primeros seis años de la Educación Secundaria Técnica o de la Educación Secundaria Agraria, el alumno obtiene un título de finalización de estudios secundarios. Se establece que acreditando los siete años de la Educación Secundaria Técnica o de la Educación Secundaria Agraria, el alumno recibe el título de Técnico en el área ocupacional específica elegida.

Acerca del diseño curricular de la educación secundaria modalidad técnico profesional [35], se estipulan las siguientes tecnicaturas:

- Técnico en Electromecánica
- Técnico en Administración de las Organizaciones
- Técnico Químico
- Técnico en Tecnología de los Alimentos
- Técnico en Electrónica
- Técnico en Informática Personal y Profesional
- Maestro Mayor de Obras
- Técnico en Aeronáutica
- Técnico Aviónico
- Técnico en Automotores
- Técnico en Servicios Turísticos
- Técnico en Multimedios
- Técnico Constructor Naval

Es de notar, que la formación en informática adquiere el grado de tecnicatura, sentando las bases para la posterior incorporación del alumno en el ámbito universitario.

La formación general y común a todas estas tecnicaturas está conformada por las siguientes materias:

- Arte
- Educación Física
- Filosofía
- Geografía
- Historia
- Inglés
- Literatura.
- Política y Ciudadanía
- Salud y Adolescencia

Además, se incluyen las siguientes materias, consideradas parte de una formación científico-tecnológica.

- Matemática
- Física
- Química
- Derechos del Trabajo
- Emprendimientos Productivos y Desarrollo Local
- Módulos relacionados con las Tecnologías de la Información y Comunicación

Como producto de una revisión de la estructura curricular de las tecnicaturas existentes, podemos indicar que sólo dos de estas tecnicaturas incluyen contenido relacionado a programación: la Tecnicatura en electrónica y la Tecnicatura en informática profesional y personal. Cabe destacar, que es escasa la información acerca de cómo se realiza la incorporación de conocimientos relacionados la programación, es decir, si bien el estudiante debe aprender conceptos básicos que abarcan desde el concepto de "variable", "estructuras de control" e incluso "escritura de páginas web", no se brinda detalle de qué metodología se usará para la incorporación de estos conceptos, con excepción de una materia de la tecnicatura en informática profesional y personal donde se incluye en el contenido curricular la "Resolución de problemas simples mediante diagramación lógica".

A continuación, a modo informativo, un extracto de las secciones relacionadas a aspectos de programación de las dos (2) tecnicaturas previamente mencionadas.

Estructura curricular: Técnico en Electrónica

Materia: LENGUAJES ELECTRÓNICOS

Módulo: perteneciente a la formación Cientifico-tecnologico (5to año)

Carga Horaria Total: 72 horas reloj

Contenidos mínimos: Lenguajes y pseudolenguajes aplicados en electrónica. (C, C++, Pascal, Delphi, Visual Basic, Borland C). Introducción al Lenguaje C: Evolución histórica de los lenguajes de programación. Historia del Lenguaje C. Introducción al ANSI C. Conceptos preliminares. Conceptos básicos de la diagramación lógica. Entornos de programación. Compiladores. Linkeadores. Maneras de compilar un programa. Introducción al ANSI C: Estructura del ANSI C. Estructura de un programa en C. Operadores (Lógicos, Aritméticos y Relacionales). Tipos de datos. Modificadores de tipo. Variables locales y globales. Constantes locales y globales. Instrucciones al pre-procesador. Concepto de Biblioteca. Biblioteca estándar y de usuario. Programación Básica en C: Introducción a la programación en lenguaje C. Entrada y salida de datos (funciones básicas). Funciones matemáticas básicas (math.h). Manejo básico de caracteres (ctype.h). Controles de flujo (parte 1): Estructuras condicionales (if – switch – else…if – ?:). Controles de flujo (parte 2): Estructuras de repetición (while – do…while – for). Manipulación de caracteres (ctype.h). Manipulación de cadenas de caracteres (string.h).

Materia: LENGUAJES ELECTRÓNICOS

Módulo: perteneciente a la formación Cientifico-tecnologico (6to año)

Carga Horaria Total: 72 horas reloj

Contenidos mínimos: Programación Avanzada en C. Registros. Arrays unidimensionales y bidimensionales. Punteros. Funciones. Funciones definidas por el usuario. Tipos de funciones. Pasaje por valor y por referencia. Archivos. Archivos de texto. Archivos binarios. Introducción a la Programación de Bajo Nivel en C. Variables registro (register). Operadores a nivel de bits.

Estructura curricular: Técnico en Informática Profesional y Personal

Materia: LABORATORIO DE PROGRAMACIÓN

Módulo: perteneciente a la formación técnico específica (4to año)

Carga Horaria Total: 72 horas reloj

Contenidos mínimos: El módulo no hace referencia a ningún lenguaje de programación específico, aunque se recomienda el trabajo sobre el ANSI C/C++.

Interpretación y resolución de problemas. Interpretación de enunciados. Identificación de datos, problema a resolver, resultados. Resolución de problemas identificando los datos, planteo y prueba de la solución. Definición de algoritmo y programa. Concepto de

compilador y enlazador. Entornos de desarrollo integrados (IDE). Concepto de código fuente, objeto y binario.

Algoritmos de resolución lineal. Algoritmos de resolución mediante métodos lineales. Aplicación del criterio top down en la resolución de problemas. Concepto de variable y constante. Asignación del tipo de dato. Diagramación lógica. Modelos y estándares de diagramación. Resolución de problemas simples mediante diagramación lógica. Concepto de Contador y Acumulador.

Tipos de datos y modificadores. Tipos de datos aplicados a la programación. Determinación del tipo de dato. Variables enteras, reales, booleanas y de caracteres. Rango de datos. Variables locales y globales. Modificadores de tipos. Palabras reservadas.

Operadores y su precedencia. Operadores aritméticos, relacionales y lógicos. Operadores unarios. Operadores a nivel de bytes y a nivel de bits. Precedencia de operadores.

Estructuras condicionales. Toma de decisiones. La estructura condicional **if**...**else**. Resolución de algoritmos en los que se apliquen estructuras condicionales. Condicionales simples y anidadas. Estructuras condicionales de selección múltiple **switch**...**case**. Resolución de problemas mediante el diseño y desarrollo de programas. Prueba de escritorio.

Estructuras de repetición. Concepto de estructura de repetición. Repetición controlada por contador y por centinela. Estructura de repetición **for**. Estructuras de repetición **while** y **do**...**while**. Condiciones de corte y salida de programa. Resolución de problemas mediante el desarrollo de algoritmos donde se apliquen estructuras condicionales y de repetición.

Diseño de programas, técnicas para la construcción, documentación y seguimiento. Clasificación de los lenguajes de programación. Selección de la herramienta adecuada según la plataforma sobre la cual se realizará la solución. Confección de la documentación. Ventajas de comentar los programas realizados. Implementación y seguimiento de la solución desarrollada. Asistencia básica al usuario.

Materia: Investigación Operativa

Módulo: perteneciente a la formación científico tecnológica (6to año)

Carga Horaria Total: 108 horas reloj

Contenidos mínimos: Esta descripción presenta aquellos contenidos que podrían desarrollarse en el transcurso de las actividades formativas. La misma no indica secuencia, será el equipo docente a cargo del módulo quien resuelva en qué momento y a través de qué actividades los desarrollará.

Teoría de grafos. Definiciones, Algoritmos, Programación por camino crítico, Números aleatorios. Algoritmos de generación y prueba, Simulación, Fundamentos de modelización (paseos aleatorios). Programación lineal, Modelo, Algoritmo Simplex. Procesos de Markoff. Teoría de Colas. Teoría de juegos, Criterios de elección de estrategias, Teoría de la Decisión. Técnicas de resolución de problemas, Análisis Directo, Análisis retrógrado,

Heurística. Modelización, Etapas (con aplicación a los puntos anteriores), Aplicación a Modelos de stock. Conflicto. Resolución. Toma de decisiones.

Materia: LABORATORIO DE PROGRAMACIÓN

Módulo: perteneciente a la formación técnico específica (6to año)

Carga Horaria Total: 72 horas reloj

Contenidos mínimos: Nuevamente, en esta materia se indica que es factible de desarrollarse con cualquier lenguaje de programación, se sugiere, así como en el año anterior, el trabajo con ANSI C/C++.

Estructuras de datos. Definición e inicialización de estructuras. Acceso a los miembros de la estructura. Funciones y estructuras. Definición de tipos. Uniones. Procesamiento de archivos. Jerarquía de datos. Archivos y flujos. Acceso secuencial. Acceso aleatorio. Archivos de texto y binarios. Introducción a las estructuras dinámicas en programación. Estructuras autorreferenciadas. Asignación dinámica de memoria. Listas. Listas enlazadas. Pilas. Colas. Árboles. Concepto de Shell. Shells y sistemas operativos. Operación de los shells y conceptos de sintaxis Variables utilizadas y establecidas por el shell. Procesamiento en segundo plano y control de procesos. Sustitución de comandos. Creación de alias de comandos. Programación de shell. Generación de archivos de inicio de shell. Scripting.

Materia: LABORATORIO DE APLICACIONES

Módulo: perteneciente a la formación técnico específica (6to año)

Carga Horaria Total: 72 horas reloj

Contenidos mínimos: Nociones de transmisión de la información. Redes de datos. Servicios de TCP/IP. Fundamentos de Internet, a nivel de hardware y software. Evolución de Internet. Internet y la World Wide Web. Análisis y comparación de los navegadores WEB más utilizados. Estándar W3C.

Introducción al HTML. Especificaciones DTD, relación con la W3C. HTML y XHTML. Estructura de un documento HTML. Etiquetas y atributos generales. Cabecera de un documento HTML. Sintaxis de las etiquetas. Atributos body. Estructuración del texto. Diseño y desarrollo de páginas WEB simples. Prueba, análisis y comparación del código con distintos navegadores WEB. Manipulación de fuentes en HTML. Etiquetas para el enlace a sitios WEB y archivos. Vínculos a imágenes, videos y sonido. Generación de listas. Mapeado de imágenes. Estructuración y atributos de las tablas. Formularios, declaración y manipulación. La etiqueta como contenedor, su utilización y atributos. Hojas de estilos en cascada (CSS). Importancia de la programación utilizando el HTML estricto. Análisis de interpretación de los navegadores. Selectores. Declaraciones múltiples. Agrupación de estilos. Herencia. Ventajas de escribir código utilizando CSS. Diseño y desarrollo de sitios WEB para determinados navegadores. Códigos CSS como facilitadores

de mantenimiento de los sitios WEB. Registros de dominios. Delegación del dominio. Servidores de nombres de dominio (DNS). Sitios WEB vs Blogs. Métodos para subir el sitio al servidor (clientes FTP). Introducción a los lenguajes de clientes. Definición de scripts. Programación de scripts. Tratamiento, definición y tipos de variables. Operadores aritméticos, relacionales y lógicos. Estructuras condicionales y de repetición. Objetos, funciones y métodos. Objetos y funciones predefinidas. Los objetos del navegador. Formularios avanzados. Concepto de cookie. Las cookies y los lenguajes de clientes. Lenguajes de servidor. Diferencias con lenguaje de cliente. Introducción al PHP. Declaración de variables, constantes y tipos de datos. Funciones básicas. Operadores aritméticos, relacionales y lógicos. Estructuras condicionales. Estructuras de repetición. Funciones de usuario, pasajes por valor y por referencia. Vectores y matrices. Manipulación de cadenas de caracteres. Bases de datos en SQL. PHP y MySQL.

Capítulo 4 – OpenBlocks

OpenBlocks [5] es la plataforma sobre la cual R.I.T.A. está construido. En este capítulo se explicará su génesis y cuáles son las facilidades que brinda.

¿Qué es OpenBlocks?

OpenBlocks es un framework distribuido por el Massachusetts Institute of Technology's Scheller Teacher Education Program (STEP) y surge como tesis de maestría en Ciencias de Computación e Ingeniería de Ricarose Vallarta Roque (año 2007).

OpenBlocks está basado en las ideas de StarLogo TNG, realizando mejoras en función de pruebas realizadas sobre el mismo. StarLogo TNG está orientado al desarrollador de aplicaciones y al diseñador de lenguajes. En el caso del desarrollador, si bien puede tener conocimientos de programación, no necesariamente podría tenerlos en un lenguaje de programación particular. En el caso de los diseñadores de lenguajes, OpenBlocks busca ayudarlos a entender las complejidades a las que se enfrenta un desarrollador como experiencia que permita mejorar el diseño.

La distribución de OpenBlocks es gratuita, bajo licencia del MIT y está a disposición pública un artículo con nociones generales de la tesis, mientras que una versión más completa se reserva para miembros del M.I.T.

El framework OpenBlocks

OpenBlocks mantuvo varias de las ideas de StarLogo TNG, en particular lo que hace a la interfaz visual, es decir la forma de los bloques y conectores, y se agregaron características en pro de mejorar la vista y usabilidad de los bloques.

OpenBlocks brinda un espacio de trabajo (workspace) donde se crea un programa mediante el *drag and drop* de bloques. Los bloques se encuentran definidos en un archivo XML de definición del lenguaje. El lado izquierdo de la imagen contiene los bloques disponibles, mientras que el lado derecho, es el panel de edición (conocido como “block canvas”) y contiene los bloques que componen nuestro programa.El panel de edición además puede estar dividido en subsecciones, que corresponden a páginas. Cada página servirá de contenedor a partes de nuestro código.

La Figura 26, extraída del artículo de presentación de la tesis OpenBlocks disponible a todo público muestra a la izquierda los bloques disponibles y a la derecha los bloques que componen un programa.

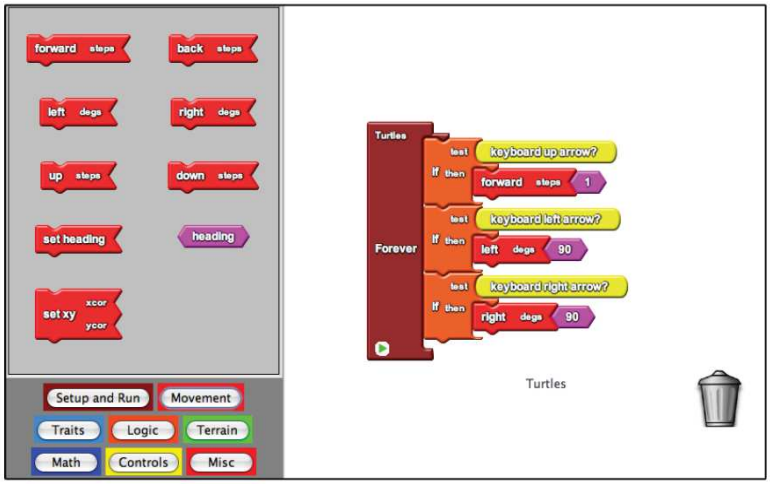


Figura 26 – Bloques en OpenBlocks

Los bloques OpenBlocks

Los bloques disponibles son administrados por "factories", de modo que su visualización puede estar agrupada por funcionalidad. El usuario toma un bloque, lo arrastra y suelta en la ubicación deseada en el área de trabajo.

Una vez que se suelta el bloque, OpenBlocks evalúa la cercanía de otros bloques y determina si puede efectuarse la conexión entre el "socket" de un bloque y el "plug" del otro bloque.

OpenBlocks provee varios tipos de conectores, a modo de ejemplo, veamos la Figura 27 donde se muestran las formas de los conectores.

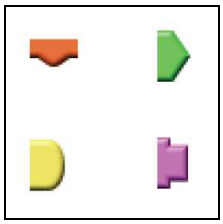


Figura 27 – Conectores en OpenBlocks

Block Drawers

Los Block Drawers son gerenciadores de bloques y pueden ser de distintos tipos:

- Factory: contiene bloques y es capaz de producir varias instancias de los mismos. También sabe como borrar un bloque cuando es elimininado del block canvas.

- Page: Es el contenedor principal que aloja nuestros bloques en el block canvas, puede existir una o más de una en el block canvas.
- Custom: en caso de querer incluir nuevos tipos de drawers. Para definir un drawer de tipo custom, es necesario modificar los archivos de configuración asociados a OpenBlocks.

Los drawers pueden ser estáticos o dinámicos. Los drawers estáticos quedan en la misma ubicación del área de trabajo durante toda la ejecución de la aplicación, mientras que los drawers dinámicos flotan sobre el block canvas y pueden ser desplazados/colapsados sobre el área de trabajo.

Características que ayudan al usuario de OpenBlocks

- Minimap: es un widget que muestra el área de trabajo de manera reducida, brindando un pantallazo de las distintas secciones del programa. El usuario de la aplicación puede posicionarse en un área del minimap y automáticamente, el block canvas centrará en pantalla el contenido seleccionado.
- Ordenamiento automático de los bloques
- Comentarios. Los bloques pueden tener asociados comentarios. El usuario de la aplicación debe hacer click con el boton derecho del mouse y seleccionar la opción “Comment Block”.

Componentes de OpenBlocks

Archivo de definición del lenguaje: Este archivo contiene información acerca de los bloques disponibles en la aplicación, y se definen sus características como etiqueta, tipo, color, etc. e información de posibles conexiones con otros bloques.

- WorkspaceController: Esta clase actúa como interfaz entre el área de trabajo y el resto de las clases que componen el framework OpenBlocks, además, el WorkspaceController es el responsable de:
    - Efectuar la carga del archivo de definición del lenguaje
    - Operatoria de carga y guardado de proyectos
- BlockLinks (Reglas de conexión): OpenBlocks provee clases que se encargan de decidir si dos bloques pueden conectarse o no, en función de si cumplen o no determinadas reglas. Por ejemplo, la clase CommandRule sabe determinar si dos bloques de tipo “command” (comandos) pueden conectarse estando uno abajo o encima del otro, mientras que la clase SocketRule se encarga de determinar si un bloque puede

conectarse con otro en función de la forma de su “socket” (se refiere al conector a la derecha del bloque) y “plug” (la forma a la izquierda del bloque) respectivamente.

- Pueden adicionarse más reglas implementando la interfaz LinkRule.
- Workspace Listener: El WorkspaceListener es una interfaz que permite escuchar eventos que ocurren en el área de trabajo, por ejemplo, cuando un bloque es agregado, removido o simplemente movido de lugar, cuando se produce una conexión o desconexión de bloque, o cuando una página es agregada, removida o renombrada en el block canvas.

Limitaciones y otras consideraciones

La principal limitación de este framework –expresado en el artículo– consiste en escribir lo correspondiente a la traducción a un lenguaje de programación ya sea compilado o interpretado.

En el artículo de presentación de OpenBlocks, se hace alusión a guías recomendadas en función de la experiencia que obtuvieron en los test de usabilidad realizados:

- Restringir el ambiente, considerando que será usado para usuarios principiantes. Es decir, que las opciones de configuración en cuanto a la vista sean limitadas, caso contrario, pueden perder el foco de atención que deberían tener en el encastre de bloques por detalles de visualización.
- Proveer constante feedback acerca de las acciones que el usuario está realizando. Por ejemplo, cuando un bloque no puede conectarse por algún motivo o si OpenBlocks estuviera ligado a un lenguaje en particular, brindar información acerca del error de compilación.
- Considerar el uso de pequeños subconjuntos de bloques. En las pruebas realizadas se encontró que muchos bloques pueden aturdir a los principiantes y hacerlos sentir inseguros acerca de que bloque tomar para lo que desean implementar.

Capítulo 5 - Robocode

R.I.T.A. tiene como aspecto motivador la construcción de un robot que pueda competir con otros. Las características generales de este robot están definidas en el framework Robocode. Este anexo trata de dar una idea del alcance de Robocode [4].

¿Qué es Robocode?

Robocode es un juego de programación donde el objetivo es programar la estrategia de un robot para competir contra otros robots en un campo de batalla. El jugador es el programador del robot, quien no tendrá influencia directa en el juego, sino que mediante el código brindará inteligencia al robot, indicando cómo comportarse y reaccionar frente a eventos ocurridos en el juego. El nombre Robocode es una abreviación para "Robot code".

El juego está diseñado para ayudar a aprender Java de una manera entretenida, donde el desafío es el estimulo para el aprendizaje. Los robots son escritos en el lenguaje de programación Java, y el juego Robocode –también escrito en Java– puede ejecutarse en cualquier sistema operativo que posea la plataforma Java, tales como Windows, Mac OS X, Linux etc.

Las batallas de Robocode ocurren en un campo de batalla, donde pequeños robots luchan hasta que sólo uno resulta victorioso en la contienda.

Breve reseña histórica

El proyecto Robocode lo inició originalmente Matthew A. Nelson, como un proyecto personal a fines del 2000 y se volvió una cuestión profesional cuando el proyecto pasó a formar parte de IBM, bajo el nombre de AlphaWorks, en Julio del 2001.

A principios del 2005, Robocode fue llevado a SourceForge [36] donde quedaba disponible como software de código abierto, en ese momento la versión de Robocode era la 1.0.7. Robocode se distribuye bajo los términos de la licencia EPL (Eclipse Public License).

En este punto, el desarrollo de Robocode se mantuvo casi detenido. Mientras tanto, la comunidad detrás de Robocode comenzó a desarrollar sus propias versiones depurando bugs y sumando valor con nuevas características en Robocode, además de las contribuciones particulares también open suorce para Robocode y las contribuciones del proyecto RobocodeNG por Flemming N. Larsen.

Viendo que nada parecía ocurrir con el proyecto Robocode en un lapso de tiempo de un año, en Julio del 2006 Larsen tomó el control del proyecto Robocode como administrador y desarrollador en SourceForge. El proyecto RobocodeNG fue cancelado, pero la variante Robocode del 2006, que contenía muchas contribuciones de la comunidad fue combinada

en la versión oficial 1.1 de Robocode. Desde entonces, se siguen realizando periódicamente nuevas versiones que incluyen contribuciones de la comunidad. A la fecha de esta presentación, la última versión corresponde a Robocode 1.7.4.1.

Requerimientos del sistema

Para instalar y ejecutar Robocode, debe tener instalado JavaTM SE (Estándar Edition) en su sistema. La actual versión de Robocode requiere al menos Java 5 (Java 1.5.0) o más reciente. Java puede ser descargado de manera gratuita y puede ejecutarse en prácticamente todos los sistemas operativos.

Antes de descargarlo e instalarlo, puede verificar si ya posee una instalación disponible desde la página: http://www.java.com/en/download/installed.jsp

Configurar el PATH

Es muy importante configurar el PATH apuntando a la carpeta “bin”, de modo de poder ejecutar la maquina virtual de Java –java.exe– desde cualquier ubicación en la línea de comandos.

Configurar el path en Windows

Agregue la ruta hasta la maquina virtual de Java configurando la variable de entorno PATH, de la siguiente manera PATH=%PATH%;JAVA_HOME\bin.

1. Encuentre el directorio donde tiene instalado Java. La ubicación default es: C:\Program Files\Java\jre6\bin
2. Copie esa ruta.
3. Haga click con el botón derecho del mouse en “My Computer” (“Mi PC”) y seleccione “Properties” (“Propiedades”). A continuación, la ventana de diálogo del sistema aparecerá. En Windows Vista, seleccione "Advanced system settings" (“Configuracion Avanzada”) en el panel lateral. En XP, seleccione el tab “Advanced” (“Avanzado”).
4. Ahora haga click en el botón ubicado en la parte inferior derecha de la ventana, etiquetado "Environmental Variables" (“Variables de Entorno”).
5. Bajo la categoría "System variables" (“Variables del sistema”) ubicado en la parte inferior, busque la variable "Path" y haga doble click sobre esta.
6. En el campo de texto "Variable value" (Valor de Variable), ubique el cursor después del último carácter de lo que ya está configurado (recuerde escribir punto y coma al final del texto original si no existe este punto y coma) y pegue la ruta hasta alcanzar el directorio local java\bin (el que copió previamente). Finalmente, a continuación de la ruta recién ingresada escriba punto y coma para finalizar la línea.

Descarga de la aplicación Robocode

A la fecha, la última versión del proyecto Robocode es la 1.7.4.1 y puede descargarse desde: http://sourceforge.net/projects/robocode/files/

Una vez descargado, la instalación se realiza desde la línea de comandos, ejecutando:

```
java -jar robocode-1.7.x.x-setup.jar
```

Esta ejecución nos guía en el proceso de instalación. Por defecto nos dejará la instalación en el directorio "robocode" de la raíz de nuestro sistema. La instalación nos deja a disposición un archivo robocode.bat para la ejecución de la aplicación.

¿Qué es un robot?

Un robot consiste de 3 partes individuales:

- Body (Cuerpo): Es quien lleva encima el arma con el radar. Los movimientos que puede hacer el cuerpo son hacia adelante, hacia atrás, hacia la izquierda o derecha.
- Gun (Arma): Montada sobre el cuerpo, es usada para disparar balas. Los movimientos que puede hacer son hacia la izquierda o derecha.
- Radar: Montado sobre el arma, es usado para "escanear" otros robots cuando se mueve. El movimiento que puede realizar es hacia la izquierda o derecha. Este radar genera "avisos o señales" cuando un robot es detectado.

  La Figura 28, extraída de http://robowiki.net/ muestra gráficamente la composición de un robot.

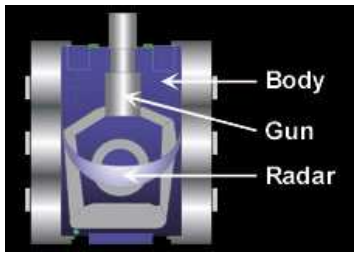


Figura 28 – Composición de un Robot

Un Robot básicamente opera de la siguiente manera:

- Se inicializa
- Busca a otros robots
- Ataca a otros robots
- Se defiende de otros robots

¿Qué facilita Robocode?

Robocode nos brinda en particular 2 herramientas necesarias para la creación y prueba de la estrategia de nuestro robot:

- Un editor de texto, básico, para escribir el comportamiento de nuestro robot. Por defecto se muestra un código básico a modo de referencia para el usuario. La Figura 29 muestra como se visualiza este sencillo editor de texto.

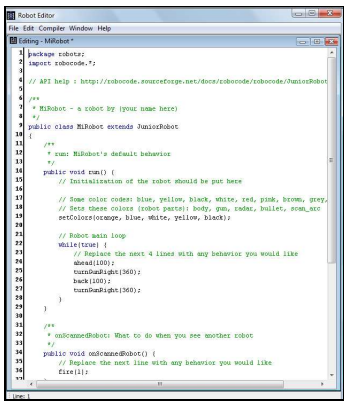


Figura 29 – Editor Robocode

- Un campo de batalla, donde nuestro robot será puesto a prueba, el cual puede visualizarse en la Figura 30.

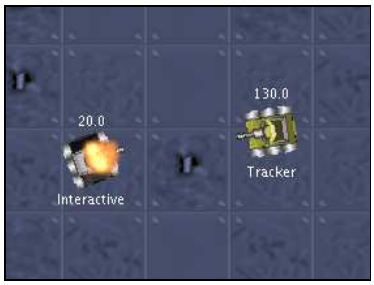


Figura 30 – Campo de batalla Robocode

Coordenadas y convenciones de dirección

El campo de batalla de Robocode tiene las siguientes consideraciones acerca del campo de batalla:

- Robocode usa el sistema de coordenadas cartesianas. Es decir la coordenada (0, 0) está ubicada abajo a la izquierda del campo de batalla.
- La dirección se fija según las agujas del reloj: 0 / 360 grados hacia el "Norte", 90 grados hacia al “Este", 180 grados hacia el “Sur", y 270 grados hacia el “Oeste".

  La Figura 31, extraída de http://robowiki.net/ muestra las convenciones acerca de las coordenadas en el campo de batalla.

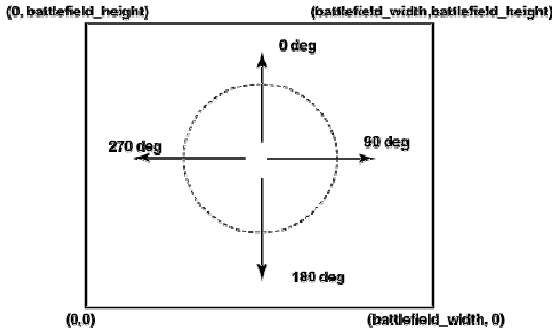


Figura 31–Sistema de Coordenadas absolutas

Robocode también tiene convenciones acerca del Bearing (ángulo relativo al cuerpo del robot)

- El rango de grado, va desde los -180 a 180 grados
- La orientación es relativa a la posición en grados a la orientación de nuestro robot
- La Figura 32, extraída de http://robowiki.net/ muestra como se consideran los ángulos relativos al cuerpo del robot.

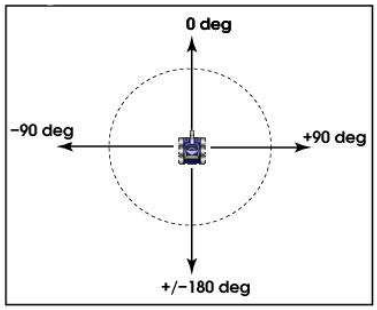


Figura 32 – Sistema de Coordenadas relativas al cuerpo del robot

La energía de un robot

Al comienzo del juego, nuestro robot –representado por un tanque– posee una determinada cantidad de energía, durante el trascurso del juego, esta cantidad de energía variará en función de las acciones que realice. Un robot sin energía no puede realizar más acciones, por lo que inevitablemente será alcanzado por las balas enemigas y se producirá su muerte.

A modo de resumen, las variaciones de la energía se deben a las siguientes situaciones:

- **Al producir un disparo.** Cuando se dispara se penaliza restando una determinada cantidad de energía dependiendo de cuanta energía haya empleado en el disparo.
- **Chocar contra un robot.** Si durante el transcurso de la batalla nuestro robot choca con algún otro robot se pierde una cierta cantidad de energía.
- **Chocar contra los muros.**
- **Recibir un impacto.** Si nuestro robot es alcanzado por una bala (enemiga o amiga) se resta una cantidad importante de energía. Esta es la mayor penalización que ocurre.
- **Utilización intensiva del radar**.
- **Tiempo de inactividad.** Si transcurre mucho tiempo y no se produce ninguna acción se sufre una penalización de energía disminuyendo a medida que el tiempo avanza.
- **Acierto sobre el enemigo.** Si una bala de nuestro robot acierta sobre un robot enemigo, nuestro robot recibe una cantidad de energía adicional.

A continuación, en detalle presentamos más consideraciones que deberían tenerse en cuenta al pensar una estrategia:

Acerca de las Balas

- Daño: 4 * poder de fuego.
    - Si poder de fuego > 1, hace un daño adicional = 2 * (power – 1)
- Velocidad: 20 – 3 * poder de fuego
- Calor generado por el arma: 1 + poder de fuego/ 5.
- No se puede disparar si el calor generado > 0.
- Todas las armas están calientes al comienzo de cada “round”.
- Poder devuelto al acertar al enemigo: 3 * poder de fuego

Acerca de las Colisiones

- Si colisiona con otro robot:
    - Cada robot recibe 0.6 de daño.
    - Si un robot se estaba alejando del punto donde ocurrió la colisión, no será detenido.
- Si colisiona con un muro:

- Los AdvancedRobots pierden abs(velocidad) * 0.5 – 1; (nunca es menor a 0).

Acerca del Incremento de energía

- Nuestro robot empieza con una cierta cantidad de energía (100)
- Acierto sobre el enemigo. Si una bala de nuestro robot acierta sobre un robot enemigo, nuestro robot recibe una cantidad de energía adicional.
- Los robots en estado "agotado" por disparar demasiado son deshabilitados (Si alcanza a otro robot, gana nuevamente energía)

Estrategias de un Robot

La estrategia de un robot debería hacer un uso óptimo de cada una de sus componentes, como el radar y las balas. Asimismo, la estrategia de movimiento permitirá no ser alcanzado por sus enemigos.

Acerca de la Estrategia de radar

La estrategia de radar se clasifica en:

- Simple. Apuntar el radar en la misma dirección que el arma.
- Circular. Mantener el radar girando constantemente.
- Tracking. Mantener el radar fijo sobre un objetivo.
- Optimal. Mover el radar de modo de escanear todos los robots tan rápido como sea posible.

Hay que tener en cuenta que escanear consume tiempo.

Acerca de la Estrategia de movimiento

La estrategia de movimiento se clasifica en:

- Movimiento en una línea recta. Fácil de predecir por enemigos.
- Movimiento en curva (circular). Evita ser alcanzado por un robot que apunta de manera estacionaria o lineal. Fácil de predecir por enemigos.
- Moverse hacia adelante o atrás, oscilante.
- Moverse en una dirección random. En general disminuye el riesgo de ser alcanzado por cualquier estrategia, aunque aumenta las probabilidades de choque.

- Anti-gravedad: mantenerse lejos de áreas peligrosas

Acerca de la Estrategia de disparo

La estrategia del disparo se clasifica en:

- **Estacionario**. Dispara hacia la actual ubicación del objetivo. Es la más simple y menos efectiva de las estrategias.
- **Linear**. Dispara donde se asume estará el objetivo.
    - Considera que se mueve a una velocidad constante en linea recta.
    - Efectivo cuando el objetivo está cerca.
    - Los movimientos de un robot pueden ser aproximados en una linea recta por un periodo muy corto de tiempo. Se vuelven poco confiables rapidamente.
- **Circular**. Dispara donde se asume estará el objetivo.
    - Asume que se mueve a una velocidad constante y a una velocidad angular constante.
    - Funciona un poco mejor que la estrategia lineal.
    - Recomendaciòn: cambiar la velocidad rápidamente, detenerse y arrancar, moverse hacia atrás y adelante rápidamente.
- **Oscilatorio**. Se asume que el robot objetivo está moviendose hacia atrás y adelante continuamente.
    - Pocos robots implementan esta estrategia.
- **Movimiento "pattern matching"**. Dispara prediciendo una posición.
    - Guarda los movimientos pasados del robot y asume que éste se moverá siguiendo ese patrón.
    - Es mejor que las estrategias anteriores
    - Consume más tiempo de procesamiento por las búsquedas exhaustivas.

Ejecución de Robocode

La organización y ejecución de una batalla en Robocode sigue los siguientes pasos:

1. La vista en pantalla es (re)pintada.

2. Todos los robots ejecutar su código hasta que se tomen medidas (y entonces hacen una pausa).
3. El tiempo es actualizado (tiempo= tiempo + 1).
4. Todas las balas se mueven y se chequea si hay colisión. Incluye la acción de disparar la bala.
5. Todos los robots se mueven (arma, radar, orientación, aceleración, velocidad, distancia, en ese orden).
6. Todos los robots escanean (y colectan mensajes de equipo).
7. Todos los robots toman nuevas medidas
8. Cada robot procesa su cola de eventos.

Información Adicional

Tiempo y distancia en Robocode

- Tiempo (t): es medido en "ticks". Cada robot tiene un turno por “tick”. 1 tick = 1 turno.
- Medidas de distancia: En general en pixels. Hay 2 excepciones:
- Todas las distancias tienen precision en numeros reales, es decir, es posible moverse una fraccion de pixel
- Robocode escala el campo de batalla para encajar en la pantalla.

Física del movimiento del Robot

- Aceleracion (**a**):
  - Los Robots aceleran a razón de 1 pixel/turno
  - Los Robots desaceleran a razón de 2 pixels/turno
  - Robocode determina la aceleración, basado en la distancia que el robot se quiere mover
- Ecuación de la Velocidad (**v**): v = at.
  - La velocidad nunca puede exceder 8 pixels/turno.
- Ecuación de la distancia (**d**): d = vt.

Rotación del Robot, Arma y Radar

- Máxima velocidad de rotación de un robot:
  - (10 – 0.75 * abs(velocidad)) grados/ turno.
  - A más velocidad, más lento el turno.
- Máxima velocidad de rotación de arma:
  - 20 grados/turno (sumado a la velocidad de rotación del robot)
- Máxima velocidad de rotación de radar:
  - 45 grados/turno. (sumado a velocidad de rotación de radar)

Cómo se crea un Robot en Robocode

La interfaz gráfica provista por Robocode, permite crear un Robot de la siguiente manera:

La Figura 33 muestra cómo se crea un nuevo robot. Desde el menú superior principal se debe seleccionar la opción "Robot", a continuación se desplegará un menú con la opción "Editor".

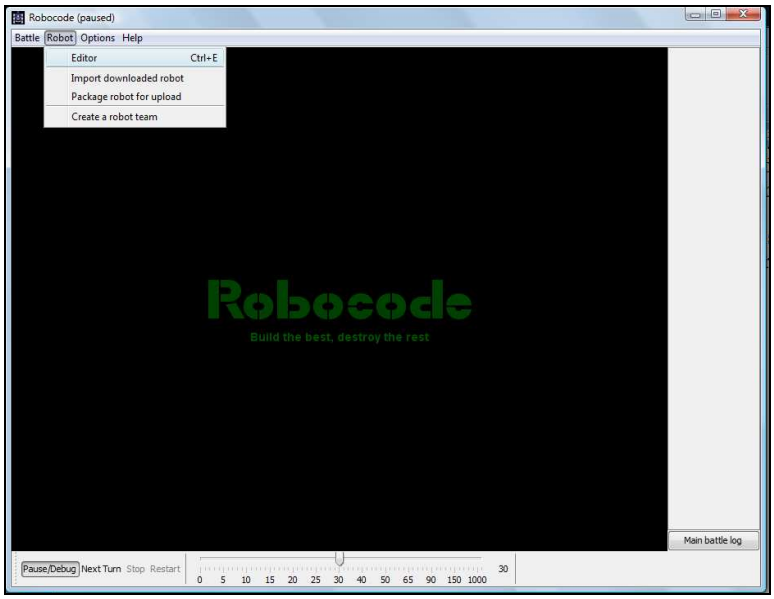


Figura 33 – Menú Robot en Robocode

La Figura 34 muestra la ventana resultante de la acción anterior. Ésta ventana conocida como "Robot Editor" nos permitirá crear un Robot básico:

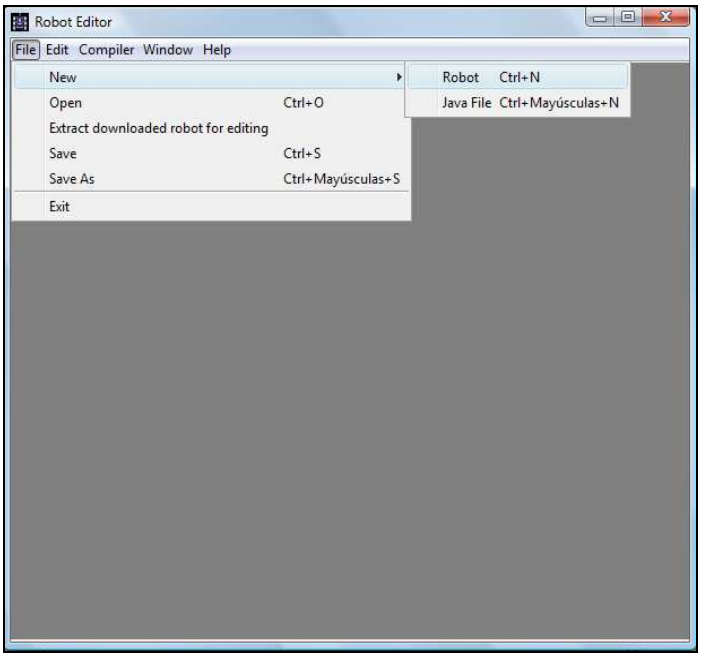


Figura 34 – Creación de un Robot en Robocode

Al hacer click en la opción de menú New > Robot, se nos mostrará una ventana donde vamos a ingresar el nombre del Robot, este nombre debe comenzar con letra mayúscula, ya que se corresponderá con una clase Java. Una vez ingresado el nombre, haremos click en el botón "Aceptar". La Figura 35 muestra como se realiza la creación del robot.

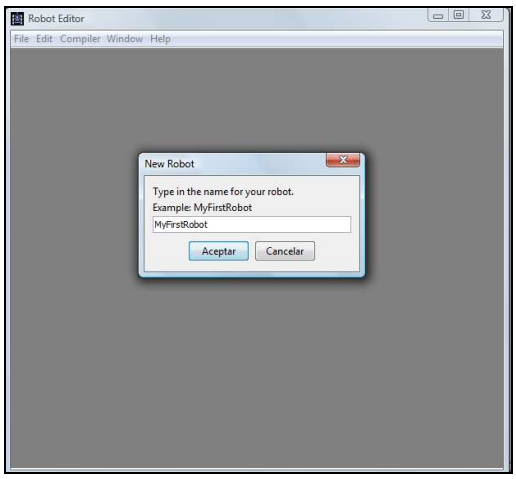


Figura 35 – Ingreso del nombre de nuestro Robot en Robocode

Una vez que hemos ingresado el nombre, se nos pide un nombre "identificatorio" que permita diferencia nuestro robot de algún otro robot ya existente y que se llame de la misma manera. De hecho, lo ideal sería que todos los robots creados por un mismo jugador pertenezcan al mismo "*paquete*". Este nombre identificatorio se escribirá en minúscula, y será el nombre de un paquete en Java. La Figura 36 muestra como se

visualiza ésta ventana de creación de paquete. Una vez completado este nombre, hacemos click en el botón “Aceptar”.

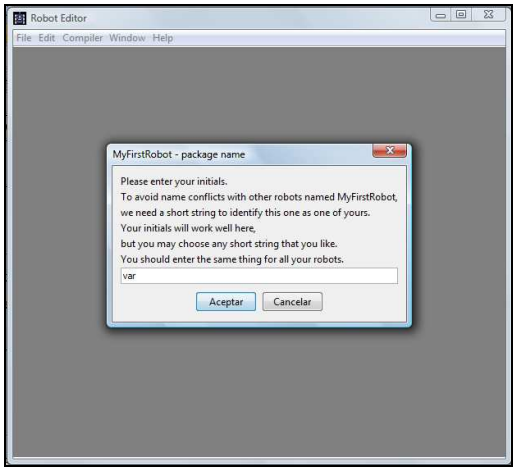


Figura 36 – Ingreso del nombre de paquete de nuestro Robot en Robocode

Una vez realizados los pasos previos, Robocode nos muestra un código básico que permite a un robot combatir, aunque sin mucha inteligencia. El código generado es el que se muestra en la Figura 37.

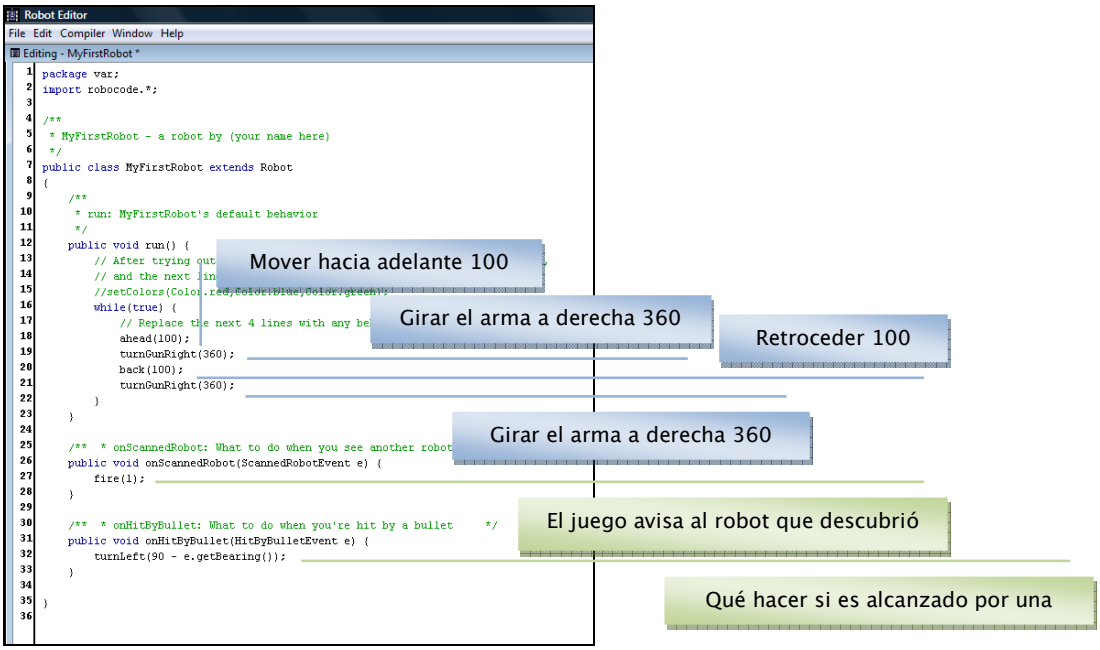


Figura 37 – Código base de un Robot en Robocode

En este punto estamos listos para “guardar” nuestro código desde la opción de menú File > Save) y luego “compilar” este mismo código desde la opción Compiler > Compile. Una vez realizado todo esto, tenemos nuestro primer robot compilado en Java.

Inicio de una batalla en Robocode

Una vez realizada la instalación, iniciar una batalla para probar el funcionamiento de Robocode es muy simple.

Primero, en la barra de menú superior debe seleccionar la opción Battle > New, como se muestra en Figura 38.

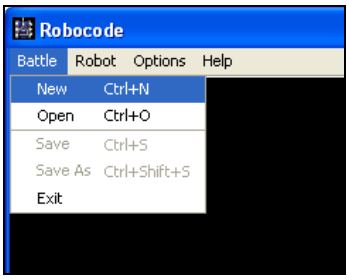


Figura 38 – Creación de una batalla en Robocode

A continuación aparecerá la pantalla "New Battle" (Nueva Batalla), donde podrá seleccionar los robots y opciones para la batalla.

La pantalla de selección de Robots, está compuesta por 2 bloques principales.

- El bloque de la izquierda que muestra los Robots disponibles organizados en "packages" (paquetes), donde una vez seleccionado el "package" se muestran los robots ubicados dentro de ese "package". Consideremos a cada package como la forma de reunir todos los robots creados por un mismo autor.
- El bloque de la derecha que muestra la selección final realizada

Para esta batalla se seleccionó MyFirstRobot, RamFire y Tracker. Se agregan estos 3 robots al campo de batalla haciendo doble click en sus nombres o seleccionando cada uno y hacienda click en el boton "Add" (agregar). La Figura 39 muestra como se realiza ésta configuración.

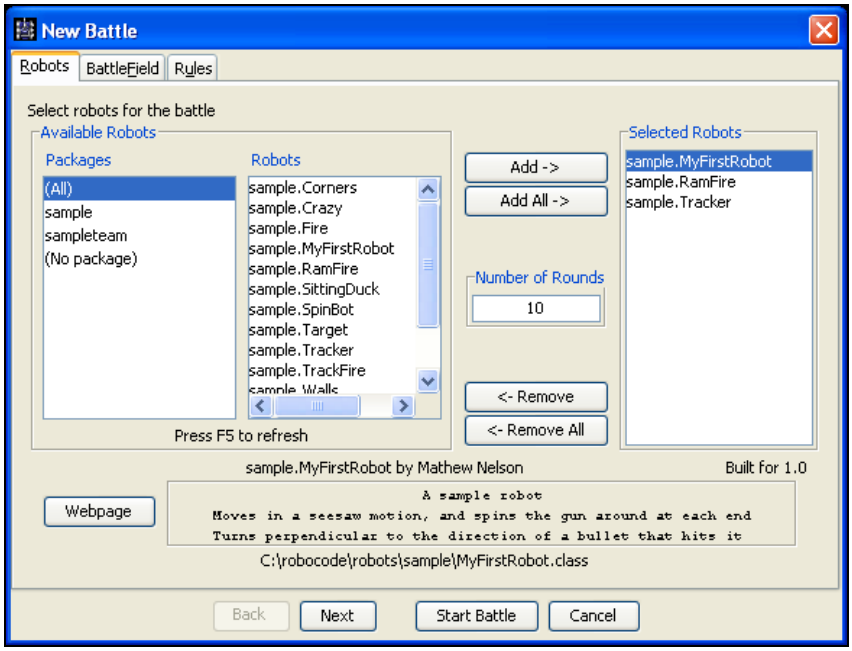


Figura 39 – Selección de los robots que competirán en una batalla Robocode

En Robocode, cada batalla consiste de un número de "rounds", por defecto 10 rounds conforman una batalla. Esta opción puede modificarse.

Finalmente hay que hacer click en el botón "Start Battle" ("Comenzar Batalla") para comenzar la batalla.

Modos de Batalla en Robocode

Las batallas se pueden plantear en 3 escenarios:

- **Uno contra uno:** En este tipo de contienda nuestro robot se enfrenta a un único robot oponente. Nuestro robot debe destruirlo antes de que el enemigo destruya a nuestro robot.
- **Mele:** Todos contra todos. En esta situación estamos rodeados de robots enemigos, donde ellos a su vez son enemigos entre sí. Esto puede considerarse un factor adicional para codificar nuestra estrategia.
- **Lucha de equipos:** Debemos armar un conjunto de N robots que lucharán de forma coordinada contra otro equipo. En este tipo de batalla se pone a prueba la colaboración y comunicación en cada equipo.

Competencias Robocode

La competencia oficial Robocode es RoboRumble [37], además es la competición más importante para los robots de Robocode, con categorías para 1v1, Melee, Teams, y Twin Duel, y subcategorias para MiniBots, MicroBots, y NanoBots (categorizados en función de la cantidad de líneas de código).

Capítulo 6 - El Proyecto R.I.T.A.

La gestación de R.I.T.A. como proyecto pasó por varias etapas, en este capítulo se incluye información referente a su evolución, y lo que corresponde a los conceptos básicos y cuáles son las opciones que brinda como producto final. Más información específica sobre los sistemas alrededor de R.I.T.A. puede encontrarse en los capítulos previos: Openblocks y Robocode.

Evolución del proyecto

El objetivo de este proyecto consistió en construir un software suficientemente amigable para permitir a los usuarios –estudiantes de escuelas técnicas con orientación en informática– a pensar en la lógica de la solución a un problema, dejando al software las cuestiones de traducción en un lenguaje específico.

Como elemento motivador –considerando la edad de la audiencia a quien está dirigida esta aplicación– se decidió que la temática de implementación fuera un complemento al framework Robocode. Robocode brinda los elementos básicos para programar estrategias de robots y además evaluar cual es la mejor estrategia mediante una batalla donde participan los robots implementados.

En un principio, la implementación del proyecto fue planteada en JavaFX[38] (versión 1.2) perteneciente a Sun Microsystems, Inc., una tecnología relativamente nueva y muy prometedora por lo descriptivo de su lenguaje, y por ende su orientación al uso no sólo por parte de programadores sino de usuarios de otros ámbitos, como diseñadores gráficos. JavaFX 1.2 permitía construir interfaces atractivas al usuario final. Se inició entonces la implementación con JavaFX 1.2.

Cabe destacar que mientras se gestaba este proyecto, Oracle Corporation adquirió Sun Microsystems, Inc., y por ende, pasaba a ser el propietario de la nueva tecnología JavaFX. En los anuncios posteriores realizados por Oracle, se remarcó la importancia de impulsar y apoyar el progreso de JavaFX a través de nuevas versiones, descartando así cualquier rumor de discontinuación del producto; sin embargo, los usuarios de JavaFX estuvimos no menos de 1 año esperando por estas nuevas versiones, hasta que finalmente se hizo el anuncio: en septiembre del 2010 Oracle decidió discontinuar el soporte para la última versión de JavaFX y optaba por quedarse con una parte del proyecto, la cual incluiría en las futuras versiones de la plataforma Java. Oracle además decidió descartar una parte del proyecto original, el scripting, que hacía particular y diferente de JavaFX (actualmente llamado proyecto Visage[39]).

En este punto tenemos un proyecto de tesis encaminándose en una tecnología obsoleta, sin soporte del fabricante y sin expectativas de que futuros informáticos puedan extenderlo funcionalmente. Considerando que al realizar un proyecto de tesis es deseable que el proyecto sea extensible, permitiendo mejoras en el futuro, se optó también por discontinuar la implementación de esta tesis en JavaFX, y usar una tecnología estándar

incluido en todas las versiones de Java: la API Swing. En definitiva, se seguía cumpliendo con los objetivos originales y se reusaba parte de lo implementado en JavaFX.

Paralelamente a todos estos acontecimientos relacionados a JavaFX, llega a mi conocimiento un framework disponible en código fuente abierto en la web y escrito en Java, generado como resultado de una tesis de maestría del MIT: "OpenBlocks[5]" que plantea una solución a la programación en bloques.

Básicamente OpenBlocks provee "bloques gráficos" para estructurar un programa, pero no un lenguaje asociado a estos bloques. Se optó por testear este software, revisar su implementación y estructuración, llegando a la conclusión que podía ser integrado como parte de la solución buscada. Obviamente, debería modificarse, completarse y ampliarse el código de OpenBlocks para adaptarlo al resultado esperado. Asimismo, se integró todo lo relativo al lenguaje resultante esperado de la compilación: código JAVA.

En este punto tenemos una tecnología de implementación definida: Swing, un framework de programación en bloques disponible para ser reutilizado y extendido: Openblocks y un elemento motivador para los usuarios: las competencias en Robocode. Esto definió un nuevo y definitivo marco de trabajo para el proyecto.

¿Qué es R.I.T.A.?

R.I.T.A. (Robot Inventor to Teach Algorithms) es una aplicación Java que integra los frameworks OpenBlocks y Robocode. Esta integración consistió en adaptar y extender las funcionalidades brindadas por Openblocks para que brinde soporte a las clases Java provistas por Robocode, así como también a estructuras Java clásicas de modo que el usuario de R.I.T.A pueda mediante Openblocks escribir la estrategia de un Robot.

El trabajo de integración demandó por un lado la investigación del framework Openblocks, ya que la documentación disponible es muy acotada y no existe una comunidad de respaldo a la misma. Por otro lado, también se estudió la viabilidad de usar el framework Robocode como parte de la solución. En Robocode un robot es un tanque de batalla. Lo que resulta atractivo de Robocode es el público al cual está orientado, principiantes con ganas de aprender a programar y por otro lado el aspecto motivador que trae consigo, dado que la construcción realizada por el principiante (estrategia de un robot), puede ponerse a prueba frente a otros robots. Luego de realizar una revisión de las características provistas por Robocode, se llegó a la conclusión de que ésta tesis permitiría la construcción de un robot en particular: JuniorRobot.

La integración consistió principalmente en adaptar los archivos de configuración de Openblocks para que soporten Java y al JuniorRobot, y también en proveer un conjunto de clases que sirvan de complemento y soporte a las clases provistas por Openblocks, de modo que se genere automáticamente código fuente Java de acuerdo a la API Robocode. También se realizaron otras tareas también importantes como adaptación y mejora sobre el funcionamiento de Openblocks, cambios en la interfaz visual Openblocks así como incorporación de nuevas opciones, en particular se destaca la que permite compilar el código Java y poner a prueba los robots generados, permitiendo al usuario seleccionar los

robots y lanzando automáticamente el entorno Robocode. Todo de una manera transparente para el usuario de la aplicación.

La Figura 40 muestra la vista de una batalla Robocode. La estrategia de los tanques está escrita empleando R.I.T.A.

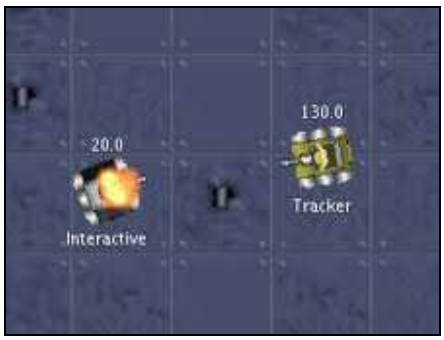
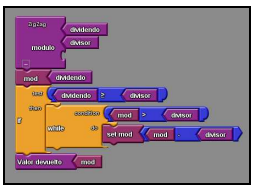
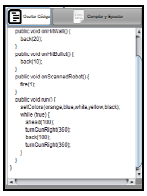


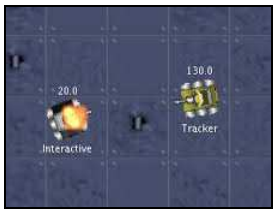
Figura 40 – Batalla en Robocode

A medida que el usuario de la aplicación planea su estrategia usando la programación en bloques, puede visualizar –si así lo desea– el código Java resultante. Una vez que el usuario termina de diseñar su estrategia de combate, puede medir su efectividad combatiendo contra otros robots. En éste punto el usuario debe visualizar el código generado, ya que la opción de poner a prueba su robot se encuentra en el mismo panel que el código fuente Java. En conclusión, R.I.T.A. permite al usuario de una manera casi transparente compilar el código Java generado y ponerlo en ejecución en el escenario brindado por Robocode.

La operatoria de R.I.T.A. para el usuario final puede simplificarse en tres (3) pasos, como se muestra en la Figura 41:







Paso 1:

El usuario estructura bloques que definen su estrategia

Paso 2:

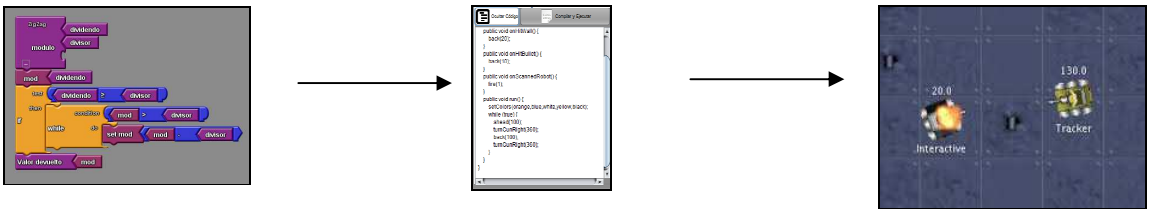
El usuario compila y ejecuta su robot

Paso 3:

El usuario observa su robot en acción

Figura 41 – Operatoria del usuario final en R.I.T.A.

A nivel de implementación, la operatoria de R.I.T.A. es un tanto más compleja, la Figura 42 muestra un esquema simplificado de lo que ocurre detrás de cámara:



Paso 1:

R.I.T.A. extiende Openblocks brindando soporte a nuevos bloques

Paso 2:

R.I.T.A. genera código Java a partir de los bloques. Genera los archivos .class y .jar

Paso 3:

R.I.T.A. dispara el ambiente de ejecución Robocode, indicándole que nuestro robot es uno de los combatientes

Figura 42 – Operatoria de R.I.T.A

Los archivos .class se generan en la carpeta correspondiente al paquete Java especificado, si no se hubiera realizado esta última acción, se ubican en un paquete generado durante la instalación, que por defecto se llama “misrobots”.

Cuando se lanza la ejecución de Robocode, R.I.T.A. le indica a Robocode información de cada robot: el nombre del paquete y el nombre del robot que deberá ser puesto en combate.

R.I.T.A. permite crear robots simples, conocidos como JuniorRobot[40]. Este tipo de robot si bien es el más simple dentro de las categorías de robots que pueden construirse con Robocode, permite poner en práctica los conceptos de programación que los alumnos deben aprender, al menos en la primera etapa del aprendizaje. De hecho, en el mismo proyecto Robocode, se recomienda a las personas interesadas en programar un robot, que empiecen tratando de programar un JuniorRobot. Un JuniorRobot mantiene información básica del estado de un robot y además reacciona a eventos que ocurran sobre él o en su entorno.

R.I.T.A. provee los siguientes tipos de bloques:

- estructuras de control (if-then / if-then-else / while / for)
- definición de variables, a las cuales se les puede dar un valor o recuperarlo
- creación de métodos con parámetros y valores de retorno
- tipos de datos Integer, Boolean y String

- variables predefinidas para un JuniorRobot
- métodos predefinidos para un JuniorRobot

La Figura 43 muestra ejemplos de bloques que representan estructuras de control

Figura 43 – Bloques que representan estructuras de control

Con estos elementos el usuario construye estructuras de bloques que corresponden a su estrategia de combate. En general, hay una relación directa entre la complejidad de la estrategia y la complejidad de la estructura de bloques resultante.

R.I.T.A. además permite:

- notificar al usuario mediante una serie de alertas (sonoras y visuales) acerca de la conexión correcta o no de los bloques
- guardar el código fuente
- compilar los archivos a en bytecode y de ser necesario notificar errores en compilación
- guardar todo el proyecto para una posterior edición
- poner a prueba la estrategia enfrentando a nuestro robot contra otros.

Dificultades en el desarrollo del proyecto R.I.T.A.

Si bien OpenBlocks constituye la base de R.I.T.A, este framework debió modificarse y ponerse a punto para poder cumplir con los objetivos planteados. Las principales dificultades encontradas fueron las siguientes:

1. Escasa documentación disponible.

Se encuentra a disposición del público en general un artículo de presentación de la tesis [5] con conceptos generales relacionados al proyecto, no se brindan detalles específicos del funcionamiento, es decir, para poder extender el funcionamiento es necesario hacer una revisión de las clases intervinientes y cómo se relacionan entre ellas. Además resulta necesario hacer un seguimiento paso a paso para poner en claro estas relaciones. Existe una versión más completa de este artículo, pero está disponible sólo para miembros del M.I.T.

Por este motivo fue necesario realizar una inversión de tiempo para entender el framework y la arquitectura del proyecto.

2. Puesta a punto de la implementación de OpenBlocks.

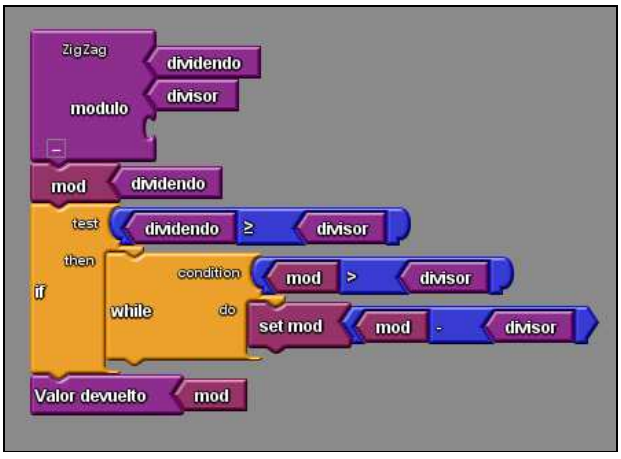
No toda la funcionalidad del framework –disponible a todo público en la web– funciona correctamente. Es de esperar que haya una versión más completa dentro entorno del MIT, dado que durante la presentación de OpenBlocks como tesis de maestría, se realizaron pruebas de campo. El aspecto más crítico fue el relacionado a la generación de nuevos "métodos" (procedimientos).

Extensiones realizadas sobre OpenBlocks

1. Métodos definidos por el usuario

R.I.T.A. permite la creación de métodos. Los métodos bajo la forma de estructura en bloques, tienen un nombre, parámetros, un cuerpo y un tipo de retorno, tal como lo tendría un método Java. Estos métodos puedan ser invocados desde otros bloques. R.I.T.A. mantiene consistencia en las llamadas a estos métodos, por ejemplo, al eliminar un método de área de trabajo[1], R.I.T.A. elimina del área de trabajo todas las referencias al mismo, ya que de otro modo resultaría en código inconsistente. Si bien OpenBlocks brinda una estructura similar bajo el nombre de "procedure", en R.I.T.A. se agregó el código necesario para su correcto funcionamiento.

La Figura 44 muestra como se visualiza un método en R.I.T.A. En este ejemplo el método se llama módulo, el cual realiza una serie de operaciones para finalmente devolver un valor.



[1] El área de trabajo constituye la sección de ´R.I.T.A. donde se ubican los bloques que constituyen la estrategia del robot, es decir, los bloques disponibles son arrastrados usando "drag-and-drop" hacia ésta sección.

Figura 44 – Ejemplo de un método en R.I.T.A.

2. Parámetros polimórficos.

Los métodos en R.I.T.A pueden tener uno o varios parámetros y de distintos tipos. Para permitir que los usuarios puedan insertar un parámetro de tipo Integer, Boolean o String, se usa un tipo de conector especial para los bloques, llamado polimórfico. Si bien OpenBlocks ya brinda una definición de este tipo de parámetro se amplió su funcionalidad.

Los parámetros polimórficos de métodos no permiten nombres repetidos y ante un cambio –por ejemplo de nombre o tipo- se actualizan las referencias desde otros bloques para mantener la consistencia.

La Figura 45 muestra un método con dos parámetros: el primero un conector de tipo "String", y el segundo con forma polimórfica

Figura 45 – Método con Parámetro. Si bien el conector del valor "texto" es el mismo, R.I.T.A. valida e informa del error al usuario final.

Adicionalmente, en R.I.T.A. se dio tratamiento especial a los bloques que representan valores de un determinado tipo de dato simple frente a los bloques que representan parámetros de un tipo de dato particular, ya que el usuario podría querer definir un método que recibe un parámetros de tipo String -con conector rectangular- y usar para esta definición un bloque que representa de valor de tipo String, cuando en realidad lo que corresponde es un bloque que representa al parámetro de tipo String. Para evitar este tipo de ambigüedades, se agregó la verificación sobre los bloques que se pretenden conectar, de modo que si el usuario busca definir un método con parámetros, sólo pueda conectar bloques que representan parámetros y no otros que puedan llegar a tener la misma forma en el conector. El usuario además recibe una notificación en pantalla acerca del error. La Figura 46 muestra cómo se visualiza esta situación.

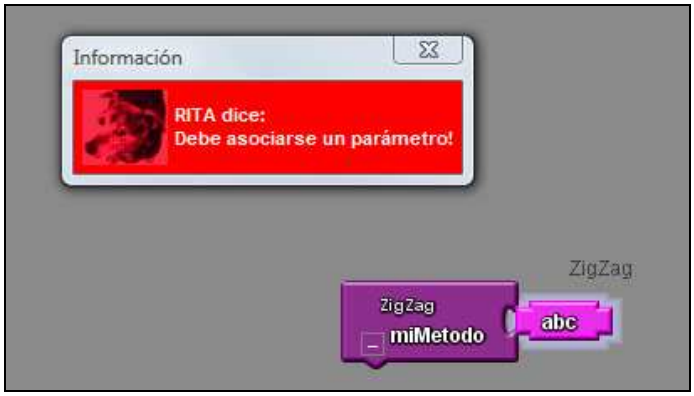


Figura 46 – Mensaje de error en R.I.T.A. cuando se pretende asociar un valor "texto" como parámetro a un método.

3. Chequeos en los tipos de retorno.

Los métodos de un JuniorRobot no poseen valor de retorno o lo que correspondería en terminología Java, devuelven "void". Los métodos que crea un usuario de R.I.T.A. pueden devolver un valor String, Integer, Boolean o void.

En el caso de los métodos definidos por el usuario, R.I.T.A verifica que sólo se permita un tipo de valor de retorno, notificando al usuario cuando incurre en inconsistencias. Adicionalmente ante un cambio en el tipo del valor de retorno del método, todas las referencias existentes (llamados al método) desde otros bloques son actualizadas, es decir, considerando que el tipo de retorno cambió, en caso de existir conexiones, se desconectan los bloques para que el usuario vuelva a reubicarlos. Para hacer más notoria esta desconexión, se emite un sonido que funciona como un feedback de la aplicación y que ayuda al usuario a visualizar que "algo" se rompió en la estructura de bloques.

La Figura 44 mostraba un método bien formado con un tipo de retorno. La Figura 47 muestra lo que ocurre cuando en un mismo método queremos indicar dos posibles tipos de valores de retorno. En el ejemplo se pretende devolver un valor numérico, y siguiendo otro flujo de ejecución un valor verdadero/false, esto genera un error el cual es notificado por R.I.T.A al usuario.

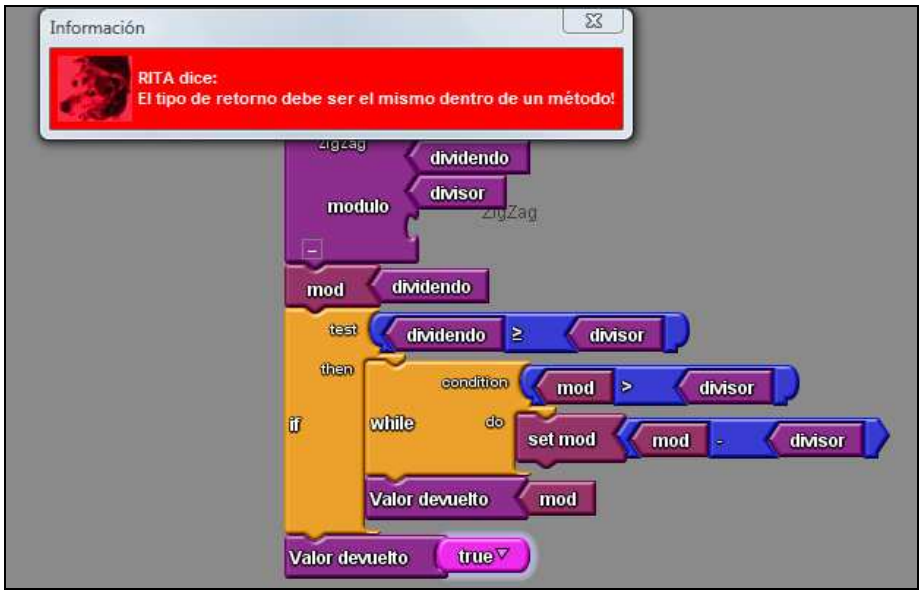


Figura 47 – Mensaje de error cuando se pretende escribir un método con 2 tipos de dato de retorno distinto.

4. **Chequeos en los nombres de ciertos elementos.** R.I.T.A. verifica que si hay dos componentes del mismo tipo, no se encuentren identificadas con el mismo nombre. Por ejemplo, en el caso de los bloques que representan parámetros no se permite el cambio en caso que ya exista otro parámetro con ese nombre en el mismo método. Tampoco se permite que el usuario arrastre 2 bloques que representan métodos de un JuniorRobot hacia el área de trabajo, ya que esto implicaría tener métodos duplicados. En éste caso particular R.I.T.A. muestra un cuadro de advertencia y no permite la incorporación del bloque al área de trabajo.

5. **Sonidos asociados a eventos.**

Se generan distintos sonidos en función de la ocurrencia de eventos como:

- la conexión básica de bloques
- la desconexión básica de bloques
- desconexión de invocaciones a métodos por cambios en la definición del método
- errores cuando dos bloques no pueden conectarse por no coincidir los tipos de conectores.
- Eliminación de elementos del área de trabajo

De esta manera, se le hace notar al usuario acerca de los cambios que ocurren en el entorno.

6. **Mensajes de Información y mensajes de Error**

R.I.T.A. muestra mensajes informativos, con fondo de color amarillo en caso de advertencias al usuario. Por otro lado R.I.T.A. muestra mensajes con fondo de color rojo cuando el usuario comete equivocaciones, ya sea conectando bloques que por la forma no pueden ser conectados, o al querer usar una variable que está definida fuera de alcance.

La Figura 48 muestra como se visualizan los mensajes de error en R.I.T.A.

Figura 48 – Un mensaje de error en R.I.T.A. se muestra como una ventana de diálogo con fondo rojo.

7. Opciones de menú sobre el proyecto

Las opciones de menú se activan o desactivan en función del estado actual de creación del robot. El usuario puede:

- guardar el área de trabajo. R.I.T.A almacena toda el área de trabajo en un archivo en formato XML. Esta opción estaba provista parcialmente por OpenBlocks, el cual provee un archivo dtd y código que recorre el área de trabajo recolectando información de los bloques. R.I.T.A hizo algunas correcciones al DTD e incorporaciones, modificaciones y validaciones al momento de generar el XML.
- Abrir proyectos iniciados para seguir trabajando en el mismo.

8. Opciones de menú de edición

El usuario puede deshacer los cambios realizados. OpenBlocks bindaba parte de esta implementación mediante el patrón Memento. En la implementación de OpenBlocks sólo se restauraba la ubicación original de los bloques. En R.I.T.A. se restaura la ubicación y las conexiones que existían antes de realizar los cambios.

9. Archivo de configuración de bloques

OpenBlocks provee un archivo de configuración de bloques básico. Este archivo fue adaptado en R.I.T.A para soportar la funcionalidad de un JuniorRobot. De esta forma se soportan bloques que representan los miembros (atributos y métodos) de un JuniorRobot. Se agregaron algunos bloques propios de Java como el “print” que realiza un “System.out.println” (impresión en la consola del Sistema Operativo) y otras estructuras más genéricas como la definición de variables con su respectivo “getter” y “setter” (métodos que leen y actualizan valores de las variables de instancia).

10. Extensión para soporte de Java

El soporte de Java se logra adaptando parte del archivo de configuración de bloques, donde se indica si un bloque puede reemplazarse por una sentencia o asociarse a un tipo en particular, y por otro lado mediante la inclusión de clases nuevas que traducen bloques

a palabras claves del lenguaje Java, como puede ser una estructura de control (if, while, for), un tipo de dato (Integer, Boolean), un operador (and/or) o bloques de comentario.

En lo que se refiere a la visualización del código fuente, se agregó un panel expandible y colapsable. R.I.T.A permite compilar el código generado en versiones .class y .jar (generando previamente el código fuente .java), por este motivo que R.I.T.A. requiere que el usuario de la aplicación tenga instalado en su máquina el Java Development Kit. Este panel también permite ejecutar el robot, es decir, poner a prueba la estrategia en el campo de batalla provisto por Robocode.

El usuario de la aplicación se ve obligado a ver el panel que muestra código fuente si quiere probar su estrategia. Esto fue hecho así de manera intencional.

R.I.T.A. trata de notificar al usuario de errores que son reconocibles antes de la compilación; sin embargo, hay algunos errores que sólo pueden descubrirse en compilación o que resultaría demasiado costoso y dificultoso de verificar previamente. En caso de ocurrir errores durante el proceso de compilación, R.I.T.A. resalta en color rojo las líneas de código que ocasionaron el error. Cuando el usuario acerca el mouse sobre la línea donde ocurrió el error, se muestra el número de línea, columna y mensaje. Éste mensaje se visualiza en idioma inglés debido a las limitaciones del compilador, ya que sólo está disponible en inglés o japonés.

Los nombres de los métodos propios de Robocode y las estructuras de control básicas se mantuvieron en inglés. En el caso de los métodos de Robocode, se tomó esta decisión dado que los nombres en español resultarían demasiado largos, a la vez que el usuario de la aplicación podría perder de vista la correspondencia de los nombres en los bloques con el código java generado, que obligatoriamente se verá en inglés. En el caso de las palabras clave, la decisión se basó en ir familiarizando al usuario de la aplicación con palabras de uso común que encontrará en casi todos los lenguajes de programación populares.

11. Ayuda

Todos los bloques disponibles en R.I.T.A. poseen ayuda descriptiva acerca de su funcionalidad en español.

12. Configuración del nombre de paquete

Robocode solicita que las definiciones de Robots se encuentren en un paquete a modo de evitar colisión en los nombres de robots pertenecientes a distintos jugadores. Por defecto siempre hay un nombre de paquete predeterminado. Luego de la instalación viene indicado por defecto el nombre de paquete “misrobots”. R.I.T.A. permite que el usuario de la aplicación lo modifique.

13. Verificación en el nombre de robot

Mientras el usuario ingresa el nombre de su robot, R.I.T.A. verifica que empiece con mayúscula (nombre de clase) y que contenga sólo caracteres válidos por convención. Caso contrario se modifica automáticamente de modo que el nombre ingresado sea válido.

14. Elección del enemigo en el campo de batalla

R.I.T.A permite poner a prueba la estrategia en 4 niveles:

- "Yo sólo": sirve para que el alumno pueda evaluar los movimientos del robot en el que se encuentra trabajando. Esta opción es útil al principio, cuando el usuario quiere verificar que las acciones que indica mediante bloques realmente se corresponden con la ejecución en el campo de batalla.
- "Fácil": pone en el campo de batalla a nuestro robot con otro pre-seleccionado, y cuya estrategia es simple
- "Medio": pone en el campo de batalla a nuestro robot con otro pre-seleccionado, y cuya estrategia es un poco más compleja
- "Difícil": pone en el campo de batalla a nuestro robot con otro previamente seleccionado, y cuya estrategia es más compleja
- "Elegir a mis enemigos": Permite al usuario elegir quiénes serán sus enemigos en el campo de batalla. R.I.T.A. busca en la carpeta "robots" (y todas sus subcarpetas) todos los robots disponibles. Esta opción resulta útil cuando tenemos varios alumnos trabajando cada uno en un robot diferente, y una vez finalizados quieren ponerlos a todos juntos en batalla. Simplemente sería necesario que ubiquen a sus robots en la carpeta "robots" de la instalación

15. Integración con Robocode

R.I.T.A. usa al framework Robocode para poder compilar los robots creados por el usuario final. Una vez que obtenemos la compilación de nuestro robot, R.I.T.A. dispara la ejecución de Robocode y pone a prueba nuestro robot en el campo de batalla.

16. Idioma del entorno R.I.T.A. (internacionalización)

El entorno de R.I.T.A. se encuentra en español, sin embargo queda abierta la posibilidad de ser extendido a otros idiomas, ya que puede ser customizado para soportarlos (internacionalización).

17. Instalación

La distribución de R.I.T.A. consiste en dos archivos, un archivo install.bat y otro RitaSetup.jar. El archivo install.bat es el instalador de R.I.T.A. El instalador de R.I.T.A. fue generado usando la herramienta de código abierto IzPack.

¿Qué es un Junior Robot de Robocode?

R.I.T.A. permite la creación de robots del tipo JuniorRobot[40], en la jerga Java nos referimos a la creación de subclases que extiendan/hereden de la clase JuniorRobot.

La clase JuniorRobot permite la creación de un robot simple, y está recomendada para programadores novatos.

La clase JuniorRobot está conformada por los siguientes miembros:

- atributos, accesibles desde cualquier otra clase (es decir, de alcance público) y no mediante los métodos "getter" y "setter" que comúnmente se usan por convención y buenas prácticas de programación, pero que sin embargo, podrían incrementar la dificultad para un programador novato.
- Los tipos asociados a estos atributos son los primitivos "int", “double” y "boolean" y atributos de color en formato RGB.
- métodos con parámetros sin valor de retorno (devuelven void) y que pueden invocarse para realizar un movimiento o una acción
- métodos que se ejecutan cuando ocurre un evento sobre el robot. No devuelven valor de retorno y deberían sobrescribirse.

R.I.T.A. conserva los tipos “Integer”, “Boolean” y “String”, estos existían en OpenBlocks bajo los nombres: number, boolean y string respectivamente. Hay un subgrupo pequeño de atributos relativo a colores que no fue incorporado para permitir el uso de clases ya provistas por Java, como es el caso de la clase Color.

Los nombres de los miembros de la clase JuniorRobot se mantuvieron en su idioma original, el inglés. La ventaja resultante son nombres de atributos y métodos más cortos, además de familiarizar al usuario con ROBOCODE en caso que en un futuro quisiera seguir programando otros tipos de robots, en otro entorno y una vez que deje de ser considerado un programador novato. Como ayuda, cuando el usuario posiciona el mouse sobre cualquier bloque, siempre se brinda una descripción de la funcionalidad en idioma español.

Proceso de instalación de R.I.T.A.

La instalación de R.I.T.A es muy sencilla. El instalador lo guiará durante el proceso, y en ocho pasos R.I.T.A. queda disponible en su equipo. Las figuras a continuación muestran éste proceso de instalación.

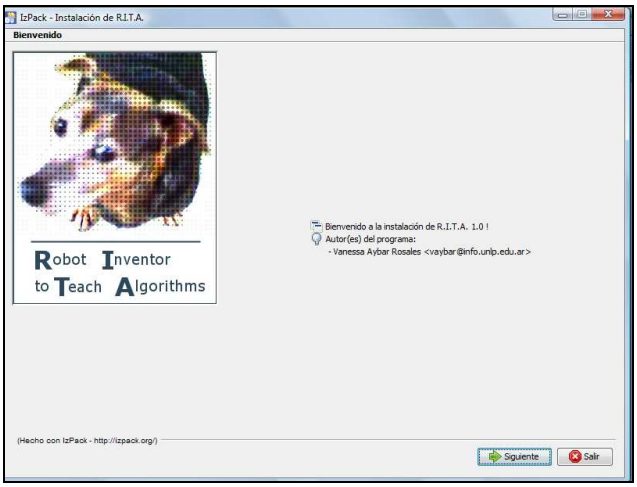


Figura 49 – Pantalla de Bienvenida a la instalación de R.I.T.A.

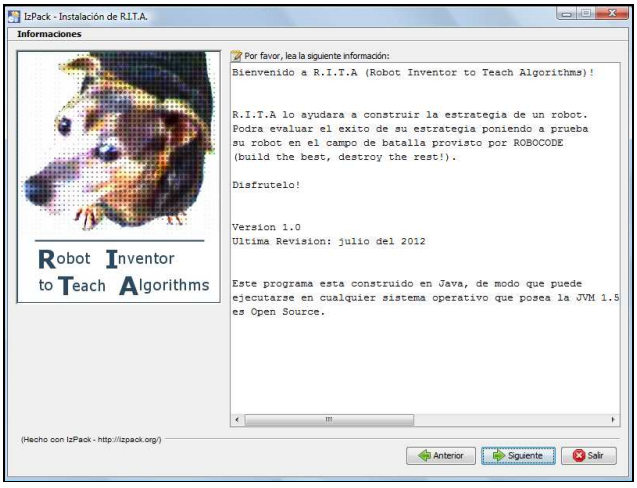


Figura 50 – Sección informativa de la instalación

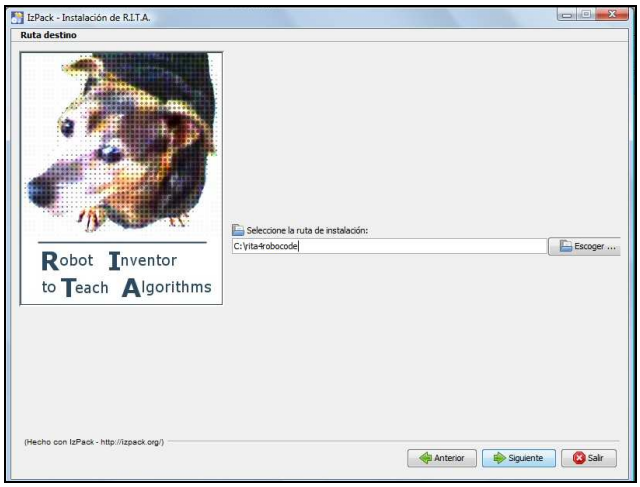


Figura 51 – Selección del directorio por default. Para evitar problemas con los permisos de carpetas en el caso de Windows Seven, se optó por dejar por defecto C:\rita4robocode

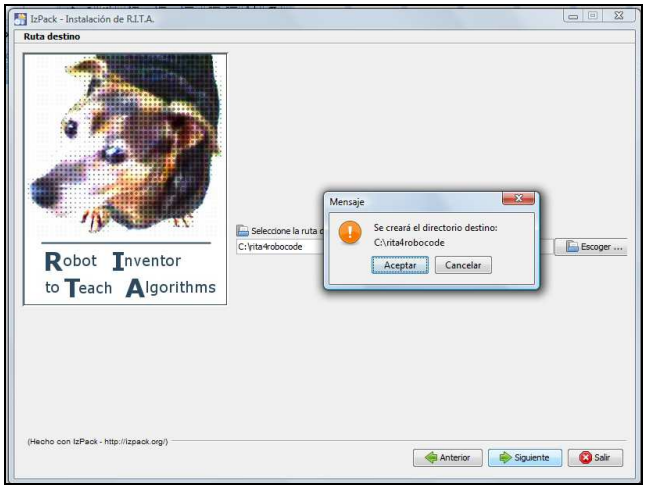


Figura 52 – Confirmación de la creación de la carpeta

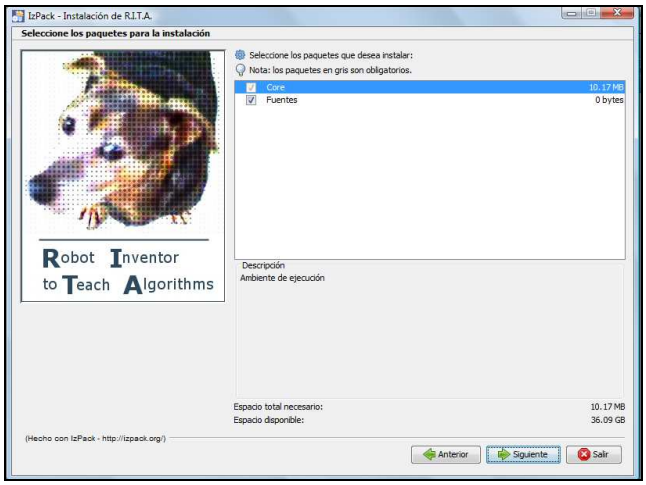


Figura 53 – Selección de las componentes a instalar

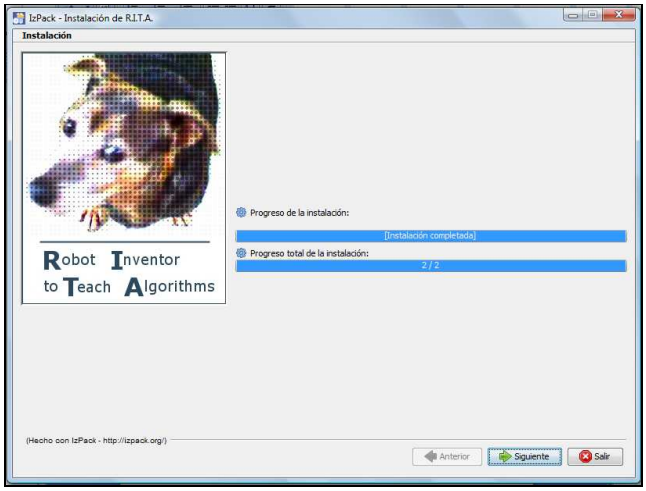


Figura 54 – Vista de progreso de la instalación

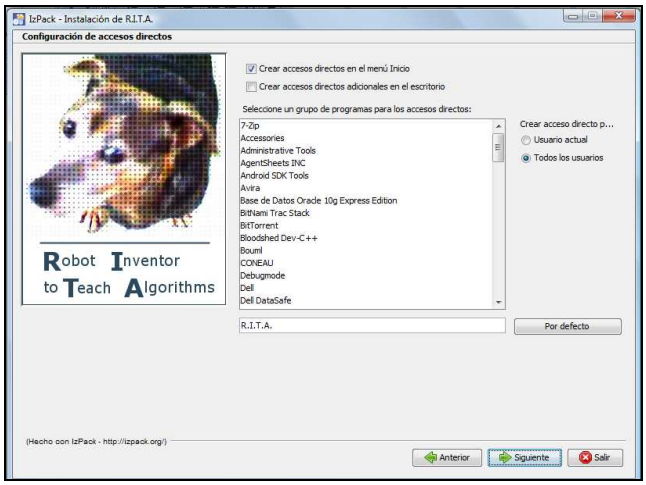


Figura 55 – Selección del grupo de programa donde se ubicará R.I.T.A.

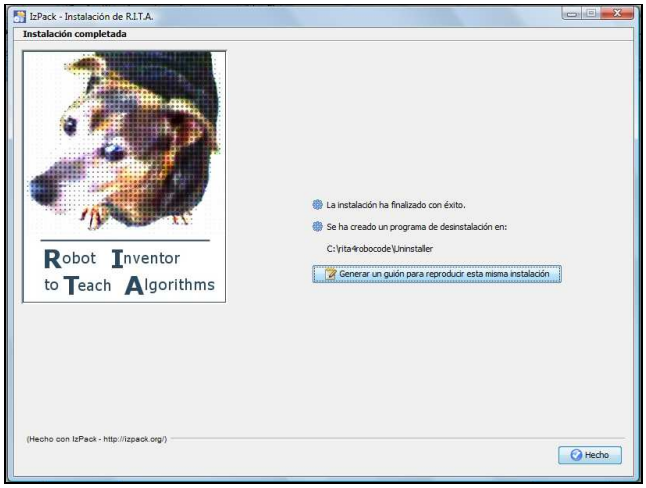


Figura 56 – Pantalla de finalización de la instalación

Una vez finalizados estos pasos, R.I.T.A. queda disponible en su equipo.

Uso de R.I.T.A

Una vez instalada R.I.T.A. en el equipo, podrá ser accedida a través del acceso directo generado en el menú de Programas.

La primera vez que R.I.T.A. es ejecutada, se muestra una ventana de advertencia acerca de una configuración básica a realizar, se solicitará el ingreso de la carpeta donde se encuentra instalado en Java Development Kit (JDK). Esta configuración es necesaria ya que

el JDK se requiere para compilar el código de los robots. La Figura 57 muestra ésta ventana de advertencia de configuración.

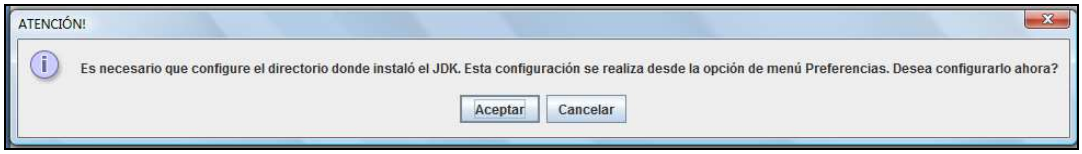


Figura 57 – Verificación de instalación de JDK

Una vez que se le indicó "Aceptar", se muestra una ventana que permite ingresar el nombre de grupo (que en la jerga Java corresponde al nombre de paquete), el directorio donde se encuentra instalado el JDK y el nivel de preparación de los adversarios cuando ponga a prueba el robot que construya utilizando R.I.T.A. La Figura 58 muestra ésta ventana conocida como la ventana de Preferencias en R.I.T.A.

Figura 58 – Ventana de configuración de preferencias en R.I.T.A.

Si el usuario lo desea puede realizar esta configuración más adelante, accediendo desde la opción Preferencias de R.I.T.A., pero no podrá ejecutar sus robots hasta que no esté configurado.

Una vez solicitada esta configuración básica, aparece una ventana solicitando el ingreso del nombre de Robot a construir. El nombre ingresado debería comenzar con una letra mayúscula, ya que el robot será una clase Java. De no ser ingresado un nombre que empiece con una letra mayúscula, se corregirá automáticamente. La Figura 59 muestra la ventana que permite el ingreso del nombre del robot.

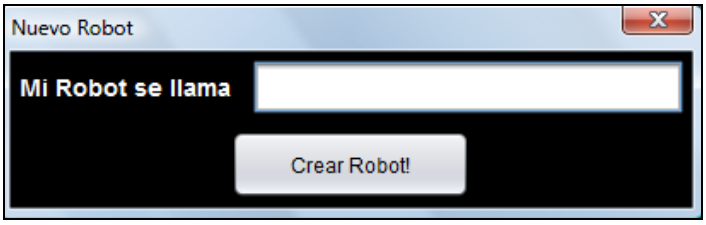


Figura 59 – Ventana de ingreso de nombre del robot

Una vez que el usuario hizo click en "Crear Robot!", se mostrará la pantalla principal de R.I.T.A.

La Figura 60 muestra como se visualiza la pantalla principal de R.I.T.A., la cual contiene varias componentes, principalmente está formado por 2 secciones principales:

- A la izquierda una sección de bloques
- A la derecha el área de trabajo principal que contiene el código del Robot

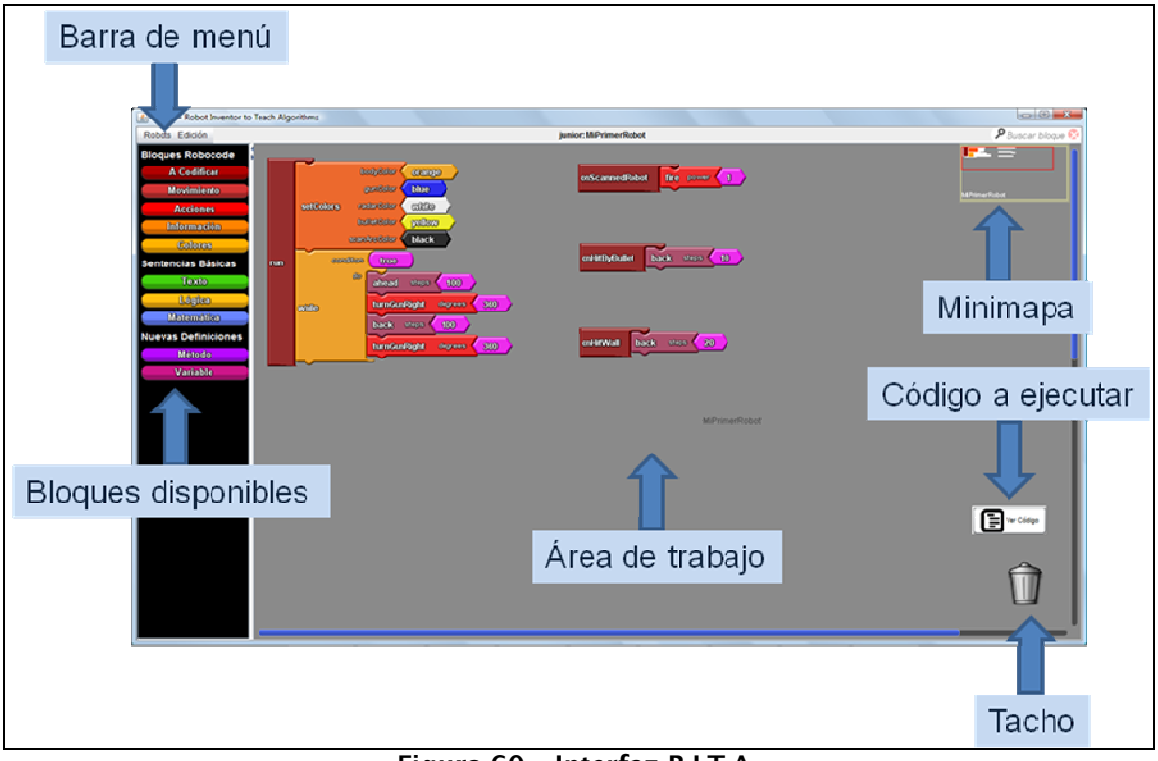


Figura 60 – Interfaz R.I.T.A.

A continuación una breve descripción de cada una de las componentes:

La **Barra de menú** brinda opciones básicas para creación de robots y guardarlos

Permite crear un nuevo robot, guardarlo con formato XML (descriptor del proyecto) y como archivo .java o abrir un archivo ya existente. La Figura 61 muestra como se despliega éste menú.

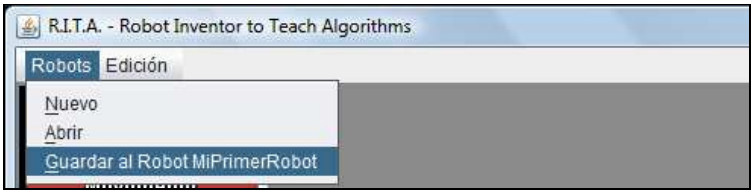


Figura 61 – Menú Robots en R.I.T.A.

La Figura 62 muestra la opción de **Búsqueda**, ubicada en la esquina superior derecha de la aplicación y que permite buscar bloques en la barra izquierda de bloques disponibles y en el área de trabajo.

Figura 62 – Opción de búsqueda de bloques

La Figura 63, muestra la sección de la izquierda de la aplicación, que contiene los **bloques básicos** para construir la estrategia del robot. Estos se encuentran agrupados en 3 subgrupos:

- Bloques Robocode: contiene métodos e información que brinda la clase JuniorRobot
- Sentencias Básicas: son aquellas que normalmente son encontradas en cualquier lenguaje de programación
- Nuevas definiciones: permite definir métodos y variables propios

Figura 63 – Grupos de bloques en R.I.T.A,

Desde la sección de bloques disponibles, podemos arrastrar y arrastrar bloques hacia el **área de trabajo**.

Cuando creamos un nuevo robot, por defecto ya aparecen algunos bloques en el área de trabajo, esta idea fue tomada desde Robocode. En Robocode, cuando el usuario quiere crear un nuevo Robot, se le brinda un editor de texto con un mínimo código a modo de sugerencia y que le sirva de guía acerca de lo que debería completar.

A partir de este punto. El usuario de R.I.T.A. puede arrastrar y soltar bloques en el área de trabajo. R.I.T.A. notificará al usuario en caso de encontrar errores acerca de lo que el usuario está tratando de hacer.

La Figura 64 muestra como se visualiza un área de trabajo que contiene bloques estructurados de manera de conformar una estrategia.

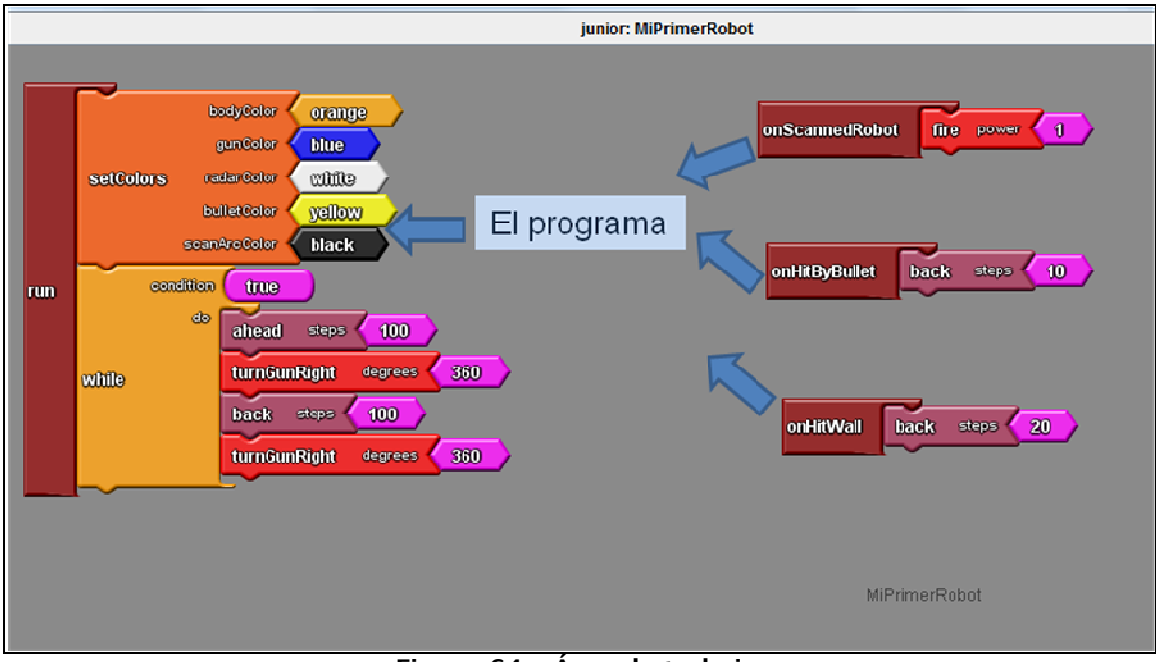


Figura 64 – Área de trabajo

Una vez que terminamos de estructurar nuestro código usando la programación en bloques, es probable que necesitemos testear como funciona nuestra estrategia. Para poder llevar esto a cabo accedermos a la opción **"Ver Código"**, que nos muestra la estructura de bloques traducida a código Java y permite "Compilar y Ejecutar" nuestra estrategia.

Si bien R.I.T.A. trata de evitar errores mientras se realiza la programación usando bloques, no es posible asegurar con un 100% de certeza que todo lo estructurado en bloques es correcto. Hay ciertas cuestiones que recién en compilación son evidentes.

Por este motivo, en caso que aún existieran algunos errores, cuando el usuario hace click en "Compilar y Ejecutar", si hay errores de compilación se le notifica al usuario mediante un mensaje de error, pero además, en la vista de código fuente se señala en rojo la línea que ocasionó el error. Cuando el usuario posa el mouse sobre la misma, se muestra en inglés la línea y columna donde ocurrió el error.

La Figura 65 muestra como se visualiza éste Panel que muestra el código Java resultante, mientras que la Figura 66 muestra como se visualizan los errores ocurridos durante el proceso de compilación.

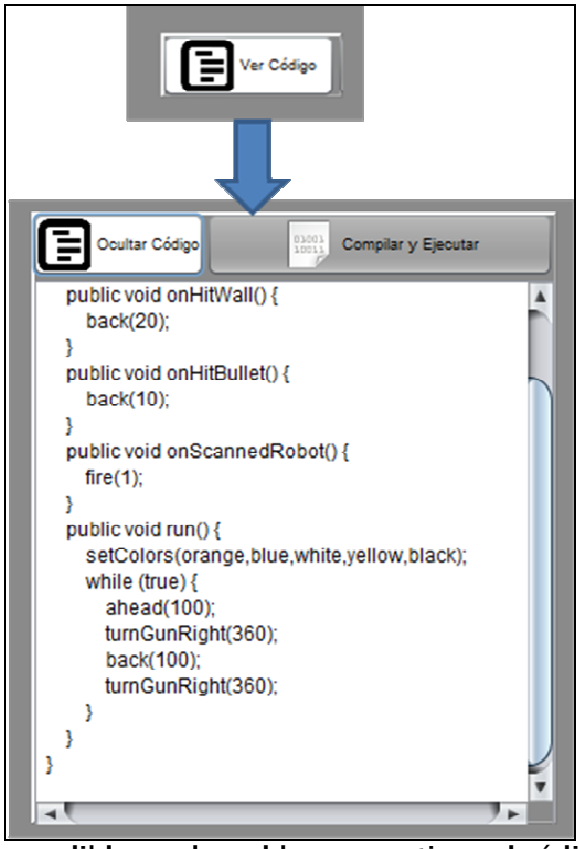


Figura 65 – Panel expandible y colapsable que contiene el código Java generado a partir de los bloques

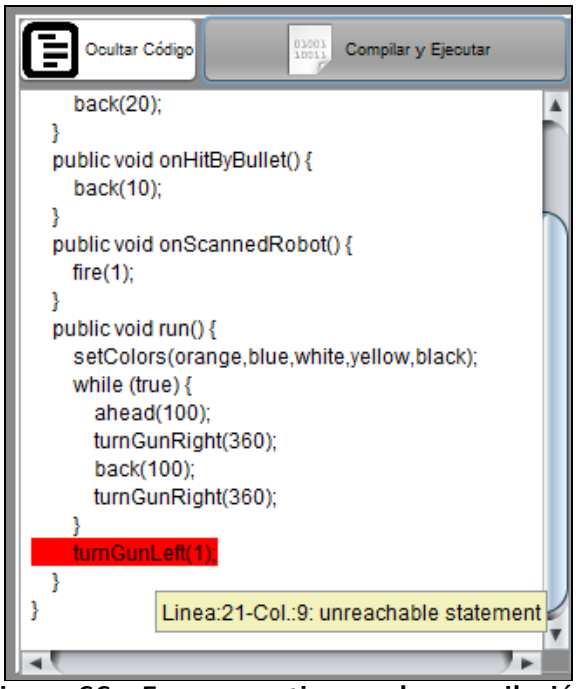


Figura 66 – Errores en tiempo de compilación

Una vez que nuestro código está libre de errores en compilación, si enviamos a ejecutar nuestra estrategia, de acuerdo a como esté configurado el nivel de dificultad de nuestro adversario, puede aparecer una ventana como la que se muestra a continuación, donde el usuario puede seleccionar cuales serán los robot adversarios en el campo de batalla.

La Figura 67 muestra como se visualiza dentro de R.I.T.A. la ventana de selección de adversarios.

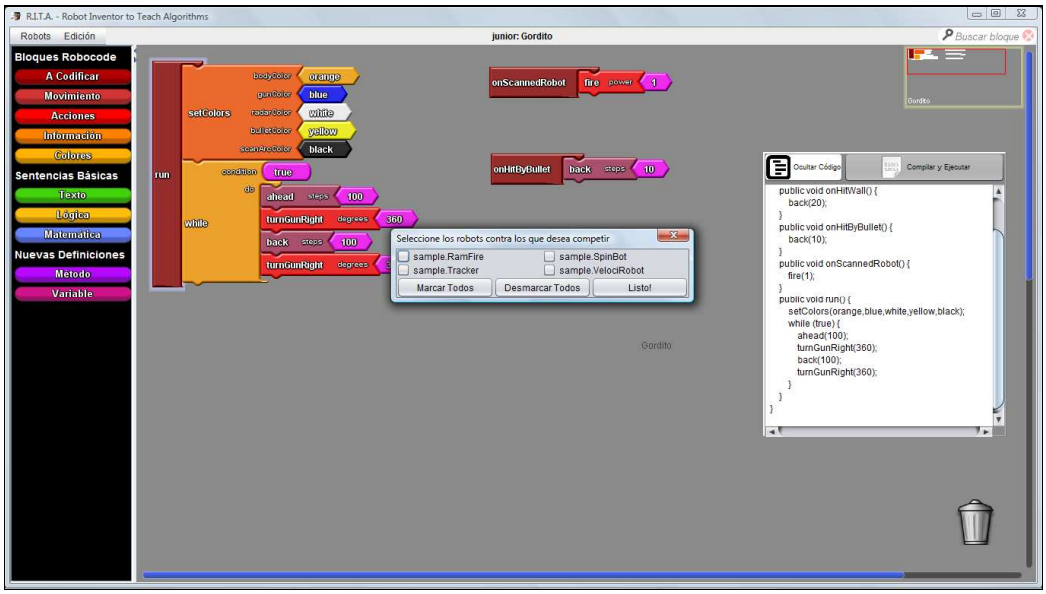


Figura 67 – Selección de robots para enfrentarse en una batalla

Una vez seleccionados los adversarios, se lanza la ejecución de nuestro robot. La ejecución de nuestra estrategia implica mostrar en pantalla el **campo de batalla Robocode** donde se encuentran ubicados nuestros adversarios seleccionados y nuestro robot. En el lado derecho del campo de batalla, el usuario puede ver el estado actual de los robots en batalla. La Figura 68 muestra como se visualizan los robots en el campo de batalla Robocode.

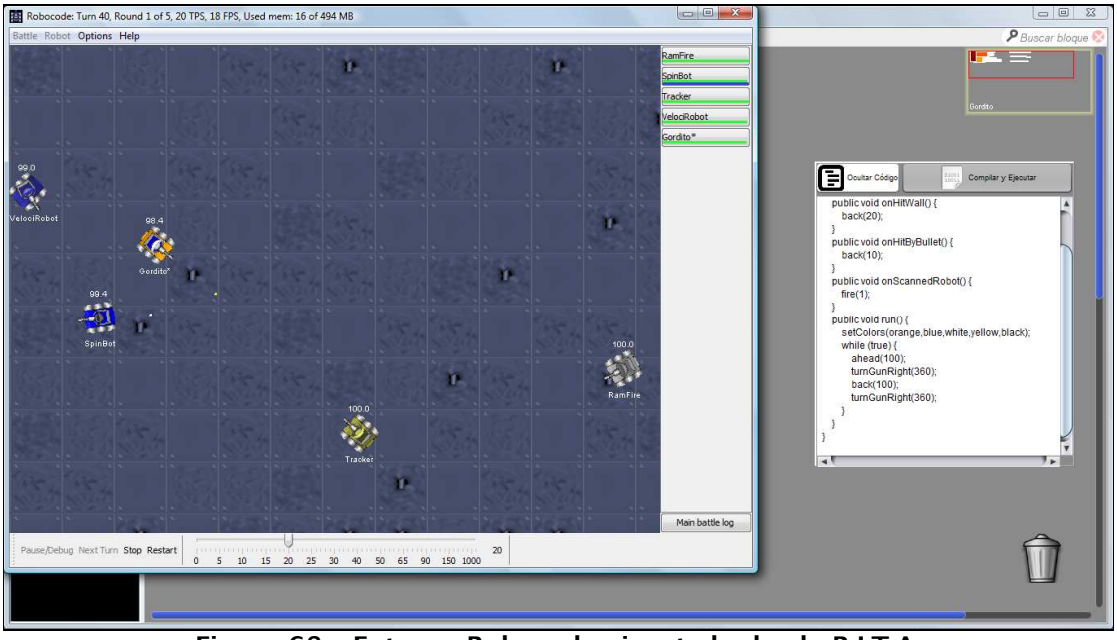


Figura 68 – Entorno Robocode ejecutado desde R.I.T.A.

Otras componentes:

Minimapa: Visualizador de todo el código, permite acceso directo a un sector en particular. La Figura 69 muestra como se visualiza el Minimapa.

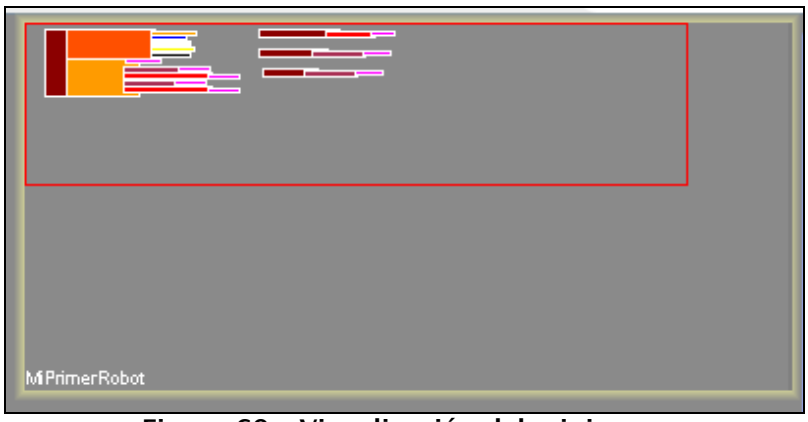


Figura 69 – Visualización del minimapa

Tacho: que nos permite eliminar componentes que descartemos de nuestra área de trabajo. La Figura 70 muestra como se visualiza el uso del tacho, cuando un bloque es arrastrado por encima del tacho, éste se muestra abierto (indicando que está en uso), y al soltar el bloque, simplemente éste desaparece y el tacho se cierra.

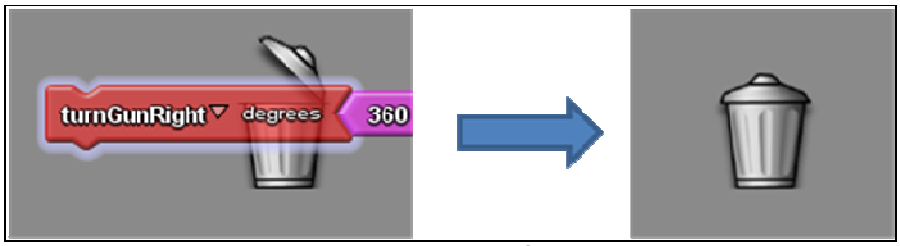


Figura 70 – Visualización del tacho

Finalmente y simplemente a modo de tener una visión general de la aplicación, la Figura 71 muestra una imagen completa de la aplicación, donde bloques estructurados en una estrategia más compleja han sido colocados en el área de trabajo.

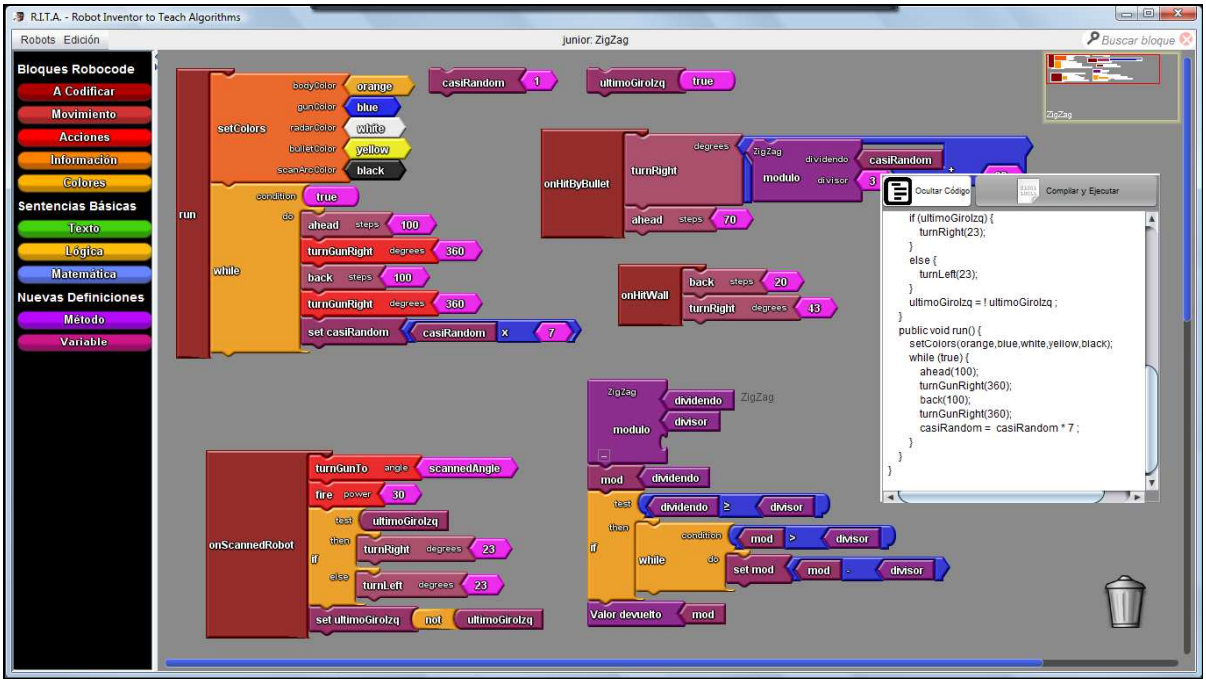


Figura 71 – Ejemplo de un programa en R.I.T.A.

Capítulo 7 – Trabajo de Campo

Encuentro con docentes de escuelas secundarias

En el marco del proyecto de extensión "Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica" (JETs)[41] [42] , el cual tienen como objetivo *"articular la educación media técnica y universitaria de la provincia de Buenos Aires en torno a un área de vacancia de interés nacional como es el desarrollo de software, mejorando y fortaleciendo a ambas instituciones"*, con fecha viernes 06 de julio del 2012 se planificó un encuentro con docentes de las siguientes escuelas:

- E.E.T. N°2 "Ing. Emilio Rebuelto" de Berisso
- E.E.T. N°5 de Berazategui

En encuentro se realizó en el aula de Postgrado de la Facultad de Informática de la UNLP y consistió de:

- Una **presentación** acerca de qué es R.I.T.A. y cómo R.I.T.A. permite escribir mediante programación en bloques código de un robot Robocode. En esta parte además se explicó la operatoria de Robocode y cuestiones a tener en cuenta acerca del funcionamiento de los robots: su composición, estrategias y movimiento en el campo de batalla.
- Un **laboratorio** donde los docentes pudieron interactuar con R.I.T.A. de modo de poder dar su impresión acerca de la herramienta

La versión de R.I.T.A. evaluada no fue la versión final, y de hecho algunas opciones se desactivaron al no estar testeadas por completo, de manera de evitar malos entendidos acerca del funcionamiento.

Al final del encuentro se les entregó una encuesta con 8 preguntas breves, la cual completaron los docentes. La Tabla 7 muestra un cuadro a modo de resumen con las respuestas obtenidas.

Cuadro resumen de la encuesta realizada a docentes de escuelas secundarias	
Pregunta	**Respuesta**
1. Acerca de la presentación inicial. ¿Considera que la información es suficiente para empezar a usar R.I.T.A.?	Todos respondieron: "SI"

2. A primera vista el ambiente de trabajo en R.I.T.A. y la cantidad de opciones disponibles en pantalla resulta…	Un 80% respondió: "Suficiente para ser entendible"
3. Cuando quiero escribir un programa en R.I.T.A. utilizando bloques, encuentro esta tarea..	Un 80% respondió: "Muy fácil de realizar"
4. ¿Se siente capaz de implementar en R.I.T.A. un plan de batalla que haya diseñado para su robot?	En general, la respuesta es "SI", sin embargo, quienes indicaron alguna dificultad hicieron referencia al conocimiento de estrategias en Robocode.
5. ¿Considera que es fácil de entender la equivalencia entre el código en bloques y el código generado en JAVA?	Un 100% respondió: "SI, es fácil de entender"
6. Respecto de la funcionalidad actual en R.I.T.A ¿Qué mejorarla?	Se observan las siguientes sugerencias: Opción de deshacer Edición del código Java Resaltar correspondencia entre bloques y código Java
7. ¿Qué cree que no tiene aún R.I.T.A y le resultaría útil a ud.?	Se planteó la incorporación de un modo de batalla donde se encuentre en el campo de batalla SÓLO el robot del alumno, de modo de poder testear sus movimientos, y la incorporación de otro modo donde puedan incorporarse más robots, dado que al ser usado en una clase, probablemente se quieran poner a prueba varios robots de los alumnos.
8. Acerca de la enseñanza de programación usando R.I.T.A comparada con escribir código JAVA directamente, opino que…	Un 100% respondió: "Es mucho más fácil usando R.I.T.A."

Tabla 7

Varias de las ideas surgidas en este encuentro fueron incorporadas en la versión final de R.I.T.A, como la opción "deshacer" (que ya se encontraba en desarrollo), y los modos de

batalla, que permiten dejar en el campo de batalla sólo el robot que se está construyendo y la opción de incorporar más robots (los robots de todos los alumnos de la clase).

El encuentro sirvió también como *test* de la aplicación, ya que durante el laboratorio se encontraron algunos *bugs* que fueron solucionados en la versión final.

Los docentes mostraron mucho interés en probar R.I.T.A. en sus respectivos cursos. A su vez, indicaron que la problemática principal que tienen hoy en día es como motivar el interés de los alumnos para el aprendizaje de programación, ya que la generación de hoy en día, a diferencia de una década atrás, necesita "comprobar" que las acciones que ellos programan, se ejecutan. Todo esto de una manera visual.

Se les preguntó a los docentes acerca de posibles inconvenientes relacionados con el idioma, ya que los métodos de Robocode, si bien tienen comentarios de ayuda en español, se encuentran escritos en el lenguaje original, el inglés. Los docentes indicaron que no creen que ese sea realmente un impedimento, y volvieron hacer hincapié en la importancia del elemento motivador de la aplicación.

Como conclusión del encuentro, R.I.T.A. cumple las expectativas de una aplicación candidata para entrenar a los alumnos en cuestiones básicas de programación, introducir algunos conceptos básicos de programación orientada a objetos como la herencia y la sobreescritura y adicionalmente, los alumnos se pueden familiarizar con la sintaxis Java.

A partir del interés surgido de este encuentro, se planificó la realización de un segundo encuentro, esta vez con alumnos con fecha 12 de septiembre (Ver Encuentro con alumnos de escuelas secundarias).

La Figura 72 y la Figura 73 muestran la página 1 y página 2 respectivamente de la encuesta entregada a los docentes.

R.I.T.A. - Encuesta

El objetivo de esta encuesta es que ud. nos dé su opinión acerca de R.I.T.A. Recuerde que R.I.T.A. está orientado a la enseñanza de programación para alumnos que realizan sus primeros pasos en ésta temática. Cualquier sugerencia de su parte resultará en una crítica constructiva acerca de la herramienta.

Para cada pregunta indiqué **SÓLO** una respuesta.

1. Acerca de la presentación inicial. ¿Considera que la información es suficiente para empezar a usar R.I.T.A.?
   - ☐ SI
   - ☐ NO, debería tener más información acerca de _____
     _____

2. A primera vista el ambiente de trabajo en R.I.T.A. y la cantidad de opciones disponibles en pantalla resulta:
   - ☐ Muy fácil de entender
   - ☐ Suficiente para ser entendible
   - ☐ Abrumador

3. Cuando quiero escribir un programa en R.I.T.A. utilizando bloques, encuentro esta tarea:
   - ☐ Muy fácil de realizar
   - ☐ Posible de realizar pero con algunas dificultades, como por ejemplo:
     _____
   - ☐ Muy dificultosa de realizar, tuve los siguientes problemas:
     _____
     _____

4. ¿Se siente capaz de implementar en R.I.T.A. un plan de batalla que haya diseñado para su robot?
   - ☐ SI
   - ☐ SI pero con dificultad, porque _____
     _____
   - ☐ NO, porque _____
     _____

Figura 72

Encuesta – Tesis de Grado: "Aplicaciones complementarias a ROBOCODE que faciliten el aprendizaje de programación en escuelas secundarias"

5. ¿Considera que es fácil de entender la equivalencia entre el código en bloques y el código generado en JAVA?

   ☐ SI, es fácil de entender

   ☐ SI, se entiende pero con dificultad, porque _____

   _____

   ☐ NO, porque _____

   _____

6. Respecto de la funcionalidad actual en R.I.T.A ¿Qué mejorarla?

   _____

   _____

7. ¿Qué cree que no tiene aún R.I.T.A y le resultaría útil a ud.?

   _____

   _____

   _____

8. Acerca de la enseñanza de programación usando R.I.T.A comparada con escribir código JAVA directamente, opino que:

   ☐ Es mucho más fácil usando R.I.T.A.

   ☐ Es un poco más fácil usando R.I.T.A.

   ☐ Opino que tiene la misma dificultad enseñar con R.I.T.A. que con Eclipse para JAVA.

   ☐ Es un poco más fácil escribiendo el código JAVA directamente en Eclipse.

   ☐ Es mucho más fácil escribiendo el código JAVA directamente en Eclipse.

Muchas gracias por participar!

2

Figura 73

El personal del área de comunicación se encargó de realizar cuatro (4) videos con la presentación de R.I.T.A. realizada frente a los docentes. Estos videos se encuentran disponibles online:

- R.I.T.A. Parte 1 (**http://vimeo.com/47670260**)
- R.I.T.A. Parte 2 (**http://vimeo.com/47787689**)
- R.I.T.A. Parte 3 (**http://vimeo.com/48070481**)
- R.I.T.A. Parte 4 (**http://vimeo.com/48070480**)

Encuentro con alumnos de escuelas secundarias

Como resultado del encuentro organizado con docentes de escuelas secundarias el 06 de julio del 2012, y también en el marco del proyecto de extensión "Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica", se organizó con fecha miércoles 12 de septiembre del 2012 un encuentro con 11 alumnos de las siguientes escuelas:

- E.E.T. Nº2 "Ing. Emilio Rebuelto" de Berisso
- E.E.T. N°5 de Berazategui

Los docentes convocaron alumnos con distinto nivel de rendimiento en la escuela, y en que general tenían alguna experiencia básica en programación.

El encuentro se realizó en el aula de Postgrado de la Facultad de Informática de la UNLP, donde los alumnos y docentes fueron recibidos con un desayuno. Adicionalmente se les entregó a cada uno de los alumnos una remera con el logo del proyecto de extensión Articular Universidad – Escuela con JAVA Para fortalecer la Educación Técnica (JETs) y una carpeta con material de de trabajo sobre a R.I.T.A. y una encuesta que completarían al final del encuentro.

El encuentro consistió de:

- Una **presentación** acerca del objetivo del encuentro: la creación de robots de batalla, y cómo R.I.T.A. ayudaría a cumplimentar este objetivo. La presentación incluía un desafío o práctica individual, para corroborar que se entendían los conceptos explicados.
- Un **desafío grupal** donde los alumnos se agruparon para intercambiar conocimiento o ideas con el objetivo de crear un robot usando R.I.T.A.

  Se organizaron en tres (3) grupos de dos (2) integrantes y un (1) grupo de tres (3) integrantes. Al final del encuentro este robot sería puesto en combate frente al resto de los robots creados por sus compañeros de aula.

Los alumnos tuvieron en todo momento asistencia para evacuar todas las dudas y que puedan cumplimentar el objetivo.

Este encuentro, así como ocurrió con el encuentro docente, permitió evaluar la herramienta e inclusive tomar nota de sugerencias y *bugs* encontrados.

Cabe resaltar que ante el desafío grupal, los alumnos demostraron mucho interés en tratar de construir un robot cuya estrategia le permite ganarle al resto de los robots de sus compañeros.

Los robots que compitieron fueron:

- MatiasyCarlos

- BTCaliber50
- Veaye
- Dartix
- Ezequiel

El combate se realizó en cinco (5) rounds donde Dartix resultó ganador en tres (3) rounds.

Cabe mencionar que algunos docentes aprovecharon el encuentro para dejar instalado R.I.T.A. en algunas netbooks del plan Conectar-Igualdad [43].

La Tabla 8 muestra un cuadro a modo de resumen acerca de los resultados obtenidos en esta encuesta:

Cuadro resumen de la encuesta realizada a alumnos de escuelas secundarias	
Pregunta	**Respuesta**
1. ¿Pensás que la explicación de Vanessa alcanza para empezar a usar R.I.T.A.?	Un 100% respondió: "SI"
2. Te parece que trabajar con R.I.T.A. es	Siete (7) de los alumnos respondieron "Fácil de entender", mientras que cuatro (4) de los alumnos respondieron "Regular de entender".
3. Te resulta más fácil programar con bloques que escribiendo código (como Pascal o VB)	Un 100% respondió: "SI"
4. ¿El robot que vos diseñaste en R.I.T.A. pudo participar en una batalla? ¿Por qué?	Un 100% respondió: "SI" y en general todos concuerdan en que pudieron programarlo ya que R.I.T.A. los ayuda a evitar errores y además pudieron comprobar el funcionamiento.
5. ¿Te resultó fácil entender el código Java generado automáticamente por R.I.T.A.?	Un 100% respondió: "SI"
6. ¿Te imaginas estudiando en esta facultad?	Nueve (9) de los alumnos respondieron "SI", mientras que dos (2) de los alumnos respondieron "Puede ser".

Tabla 8

De esta manera finalizó el encuentro. Antes de que los docentes y alumnos abandonen las instalaciones de la facultad, se organizó un almuerzo que si bien fue breve, permitió continuar charlando del tema con los docentes de escuelas secundarias, quienes quedaron conformes con la experiencia. De hecho expresaron su interés en R.I.T.A. para ayudarlos en la enseñanza de algoritmos.

El personal del área de comunicación se encargó de realizar un (1) video acerca del proyecto "Articular universidad-escuela con JAVA para fortalecer la educación técnica", cuyo contenido está basado en éste encuentro con alumnos. Éste video se encuentra disponible online en **http://vimeo.com/50735513** , el personal del área de comunicación, además, se encargó de tomaron varias fotos. A continuación algunas de las fotos tomadas durante el encuentro.

Figura 74 – Alumnos y docentes preparándose antes de la explicación de R.I.T.A.

Figura 75 – Alumnos estructurando la estrategia de su robot

Figura 76 – Alumnos consultando e intercambiando ideas acerca de la estrategia del robot.

Java en Escuelas Técnicas — UNLP | FACULTAD DE INFORMÁTICA

R.I.T.A. - Encuesta

El objetivo de esta encuesta es que nos des tu opinión acerca de esta jornada en la Facultad de Informática.

1. ¿Pensas que la explicación de Vanessa alcanza para empezar a usar R.I.T.A.?
   ☐ SI ☐ NO
2. Te parece que trabajar con RITA es:
   ☐ Fácil de entender
   ☐ Regular de entender
   ☐ Difícil de entender
3. Te resulta más fácil programar con bloques que escribiendo código (como Pascal o VB):
   ☐ SI ☐ NO
4. ¿El robot que vos diseñaste en R.I.T.A. una pudo participar de una batalla ?
   ☐ SI ☐ NO

   ¿Por qué?

   _____

   _____

   _____
5. ¿Te resultó fácil entender el código JAVA generado automátican por RITA?
   ☐ SI ☐ NO
6. ¿Te imaginás estudiando en esta Facultad?
   ☐ SI ☐ NO

   _____

   _____

   _____

Muchas gracias por participar!

Figura 77 – Encuesta entregada a los alumnos al final del desafío grupal

Capítulo 8 – Conclusión

Aspectos abarcados por R.I.T.A.

El estudio de campo realizado con participación de docentes y alumnos de escuelas secundarias, permite concluir que R.I.T.A. cubre al menos los siguientes aspectos:

- **Motivación.** El aspecto motivador de una herramienta que busca introducir a alumnos en el mundo de la programación es uno de los principales factores en el éxito de la misma. Varias de las herramientas de enseñanza de programación analizadas cubren los aspectos sintácticos o generación de código, pero carecen del aspecto motivador que impulse al alumno a continuar explorando con la herramienta o inclusive salir fuera de ella para investigar un poco más. Considerando un público adolescente de escuela secundaria, la motivación puede obtenerse al construir algo que pueda sobresalir o hacerse notar sobre el resto de la clase. El sólo hecho de "ganarle al compañero" impulsa al alumno a pensar en "cómo hacerlo", en este caso lo logra perfeccionando una estrategia de ataque y defensa de robots. Esto quedó en evidencia durante el encuentro realizado con alumnos. Todos estaban concentrados tratando de entender el funcionamiento del robot y pensando cómo lograr la mejor estrategia de combate. Como factor adicional, los alumnos invitados tenían distintos niveles de capacidad demostrada en sus escuelas, y sin embargo, todos fueron capaces de crear su robot.

- **Enseñanza de la lógica de programación mediante el mecanismo de programación en bloques.** El mecanismo de programación en bloques que emplea R.I.T.A. para estructurar un programa facilita mucho la creación de código. El alumno tiene bien demarcado que bloques pueden conectarse:

    - secuencia de sentencias
    - métodos y argumentos con sus respectivos tipos
    - alcance de declaración de las variables

  La programación en bloques elimina las dificultades de enfrentar por primera vez un lenguaje de programación, permitiendo que la tarea del alumno sea más agradable y menos frustrante. Esto queda demostrado a partir de la encuesta realizada a los alumnos, quienes se sienten más confiados al no enfrentarse directamente y en una primera instancia a los problemas de sintaxis de cualquier lenguaje de programación.

- **Enseñanza de estructuras básicas de programación.** En la versión actual de R.I.T.A. se abarcan los aspectos de enseñanza de programación básica, es decir: estructuras de control, declaración de variables con tipos de datos básicos, creación de métodos, etc. Se logra que el alumno estructure una solución. Por otro lado, al traducir automáticamente a código Java permite un acercamiento a este lenguaje. No se pretende exigir al alumno que aprenda el lenguaje de programación, sino que surja el interés por aprender, de acuerdo a sus tiempos.

- **Introducción de conceptos de Programación Orientada a Objetos.** En cuanto a los conceptos de Programación Orientada a Objetos, si bien R.I.T.A. no está pensada para enseñarlos en profundidad, sí permite introducir algunos conceptos básicos como objetos, estado y comportamiento, herencia y sobreescritura. Además es extensible para incorporar más conceptos de programación orientada a objetos. Acerca de este punto, cabe resaltar que a la fecha se realizó una reunión con dos (2) alumnos que se encontrarían interesados en realizar su trabajo de tesis abarcando este aspecto. Se les comentó qué elementos podrían incorporarse para cubrir la enseñanza de Programación Orientada a Objetos.
- **Introducción de conceptos de Programación Orientada a Eventos.** Cuando el usuario escribe la estrategia de su robot, en realidad escribe las acciones que un robot realiza frente a la ocurrencia de eventos en su entorno. Por ejemplo: el usuario debe implementar que acciones realizará el robot cuando es chocado por otro robot, o cuando colisiona contra un muro.

Mejoras a futuro

Con respecto a las mejoras sobre la implementación actual de R.I.T.A. podemos indicar la incorporación de depurador. En la versión actual de R.I.T.A. no se incorporó por cuestiones de tiempo, sin embargo es extensible para incorporarse en una versión futura.

También se podría incorporar la edición de código fuente Java, sin embargo, este punto hay que evaluarlo, ya que podría resultar contraproducente, considerando que R.I.T.A. está pensado para ser sencillo y simple de usar.

Por último R.I.T.A. puede ser extendida para incorporar otros tipos de bloques, lo que permitiría crear otros tipos de robots más avanzados y por ende profundizar en la enseñanza de programación orientada a objetos. Asimismo, esto habilitaría la creación de equipos, ésta es una característica muy interesante ya que permitiría la interacción entre los robots.

Trabajo futuro

En éste punto considero a R.I.T.A. como proyecto, o producto final. En base al trabajo de campo realizado con alumnos de escuelas secundarias, los docentes que participaron de los encuentros mostraron su interés en incluir R.I.T.A. como herramienta de soporte para la enseñanza de algoritmos/programación, por lo que probablemente requerirá cierto soporte en función de las necesidades que vayan surgiendo y el feedback recibido por parte de los alumnos.

Por otro lado, como parte del proyecto “Articular universidad-escuela con JAVA para fortalecer la educación técnica” se tienen programados encuentros in situ en escuelas de

Berazategui, Mar del Plata y Berisso para los días 31 de octubre, 2 de noviembre y 7 de noviembre del 2012 respectivamente.

R.I.T.A. además forma parte de una nueva propuesta de proyecto de extensión que consiste en una continuación del proyecto "Articular universidad-escuela con JAVA para fortalecer la educación técnica" donde aporta valor agregado al proyecto Conectar-Igualdad, de modo que los alumnos cuenten con R.I.T.A. como herramienta de programación en sus respectivas netbooks.

Referencias/Bibliografía

Bibliografía

1. **Cataldi, Zulma.** *Metodología de diseño, desarrollo y evaluación de software educativo.* La Plata : s.n., 2000. ISBN 960-34-0204-2.

3. **Bruce Eckel.** *Free Electronic Book: Thinking in Java, 3rd Edition.* s.l. : MindView, Inc. ISBN 0-13-27363-5.

6. **Manber, Udi.** *Introduction to Algorithms - A Creative Approach.* s.l. : Addison Wesley Publishing Company Inc., 1989. ISBN 0-201-12037-2.

7. **Cormen, Thomas H. - Leiserson, Charles E. - Rivest, Ronald L.** *Introduction to Algorithms.* s.l. : MIT Press, 1997. ISBN 0262530910.

8. **Weiss, Mark Allen.** *Data Structure and Algorithm Analysis- Second Edition.* s.l. : Pearson Addison Wesley Publishing Company Inc., 2007. ISBN: 0-321-37013-9.

13. **Kouznetsova, Svetlana.** USING BLUEJ AND BLACKJACK TO TEACH OBJECTORIENTED DESIGN CONCEPTS IN CS1. *BlueJ.* 2007. http://www.bluej.org/papers/2007-04-kouznetsova.pdf.

32. **Senado y Cámara de Diputados.** Ley de Educación Técnico Profesional. 7 de Septiembre de 2005. http://www.me.gov.ar/doc_pdf/ley26058.pdf.

34. Decreto Nº 144/08. http://portal.educacion.gov.ar/consejo/files/2009/12/decreto_144_08.pdf.

41. **Queiruga, Claudia - Fava, Laura.** Java en Escuelas Técnicas. *Facultad de Informática - UNLP.* http://www.info.unlp.edu.ar/article/printPreview/id/481.

Sitios de referencia validados a octubre del 2012

2. Java. *http://java.com.* Oracle.

4. *Robocode - Build the best - destroy the rest!* http://robocode.sourceforge.net/.

5. **Ricarose Roque.** OpenBlocks download page. *M.I.T. Scheller Teacher Education Program.* M.I.T. http://education.mit.edu/openblocks.

9. What is Logo? *Logo Foundation.* M.I.T. http://el.media.mit.edu/logo-foundation/logo/index.html.

10. RoboMind. *RoboMind.* Universidad de Amsternam. http://www.robomind.net.

11. **Pestov, Slava.** jEdit. *jEdit Programmer's Text Editor.* http://www.jedit.org/.

12. **Kouznetsova,Svetlana.** *BlueJ - The interactive Java environment.* University of Kent. http://www.bluej.org.

14. *Eclipse.* The Eclipse Foundation. http://www.eclipse.org.

15. *STAGECAST.* Stagecast Software, Inc. http://www.stagecast.com.

16. *AgentSheets.* AgentSheets, Inc. http://www.agentsheets.com.

17. testimonials. *AgentSheets.* AgentSheets, Inc. http://www.agentsheets.com/education/testimonials/index.html.

18. *Alice.* Carnegie Mellon University. http://www.alice.org/.

19. **Dann,Wanda - Cooper, Stephen - Pausch, Randy.** Learning to program with Alice. http://www.aliceprogramming.net/overview/AlicePresentation.ppt.

20. *Asociacion Escuelas Lincoln.* http://moodle.lincoln.edu.ar:83/course/category.php?id=22.

21. **Carlisle, Martin - Wilson, Terry - Humphries, Jeff - Moore, Jason.** *RAPTOR - Flowchart interpreter.* http://raptor.martincarlisle.com.

22. **Scott, Andrew.** *ProgrAnimate - Programming Visualisation and Animation for Novices.* University of Glamorgan. http://www.comp.glam.ac.uk/pages/staff/asscott/progranimate.

23. Introduction to Quartz Composer User Guide. Apple Inc. http://developer.apple.com/library/mac/#documentation/GraphicsImaging/Conceptual/QuartzComposerUserGuide/qc_intro/qc_intro.html.

24. *E-Cell Simulation Environment 3D.* Mesoscopic Biology Project, Institute for Advanced Biosciences, Keio University. http://ecell3d.iab.keio.ac.jp/features.html.

25. *Microsoft Robotics Developer Studio.* Microsoft. http://msdn.microsoft.com/en-us/library/dd939239.aspx.

26. Projects - Logo Blocks. *Lifelong Kindergarten.* M.I.T. http://llk.media.mit.edu/projects.php?id=141.

27. **Begel, Andrew.** *LogoBlocks:A Graphical Programming Language for Interacting with the World.* http://research.microsoft.com/en-us/um/people/abegel/mit/begel-aup.pdf.

28. Introduction to Logo Blocks. M.I.T. http://llk.media.mit.edu/projects/cricket/doc/help/logoblocks/startingwithlogoblocks.htm.

29. StarLogo TNG. *Scheller Teacher Education Program.* M.I.T. http://education.mit.edu/projects/starlogo-tng.

30. Welcome to MIT App Inventor. M.I.T. http://appinventor.mit.edu.

31. Plan de Mejoras. *Instituto Nacional de Educación Tecnológica.* Ministerio de Educación - Presidencia de la Nación. [Citado el: 19 de septiembre de 2012.] http://www.inet.edu.ar/programas/planes_mejoras.html.

33. Programas - Digital Junior : Proyecto de Certificación de Conocimientos Informáticos . *Facultad Regional Buenos Aires.* U.T.N. http://www.sceu.frba.utn.edu.ar/dav/digital_infor.htm.

35. Diseño curricular de la Educación Secundaria - Modalidad Técnico Profesional. *UDOCBA.* http://www.udocba.org.ar/index.php/2011-10-24-14-51-31/normativa/item/122-diseno-curricular-de-la-educacion-secundaria-modalidad-tecnico-profesional.

36. *Sourceforge.* Geeknet, Inc. http://sourceforge.net/.

37. RoboRumble on darkcanuck.net. *RoboRumble.* http://darkcanuck.net/rumble/.

38. JavaFX - The Rich Client Platform. Oracle. http://www.oracle.com/technetwork/java/javafx/overview/index.html.

39. **Chin, Stephen.** visage - Declarative language for expressing user interfaces. http://code.google.com/p/visage/.

40. **Larsen, Flemming N. .** JuniorRobot (Robocode 1.7.4.2 API). Robocode. http://robocode.sourceforge.net/docs/robocode/robocode/JuniorRobot.html.

42. **Queiruga, Claudia - Fava, Laura.** Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica. *UNLP - Extensionistas - Publicación de Proyectos de Extensión.* UNLP, 13 de 08 de 2012. http://www.extensionistas.unlp.edu.ar/articulo/2012/8/10/nuevas_tecnologias_en_el_aula.

43. *Conectar Igualdad.* http://www.conectarigualdad.gob.ar/.